Datalink Wristapp Programmer’s Reference

John A. Toebes, VIII

© 1997 John A. Toebes, VIII

All Rights Reserved

[jtoebes@geocities.com](mailto:jtoebes@geocities.com)

## **Table of Contents**

About the DataLink 7

So, What is a Datalink? 7

There are three basic models of the Datalink 7

What programs can I load in the 150/150s? 7

Datalink Technical Details 7

Download Protocol 7

Synchronization Process 7

Sync Bits 7

Packet Format 7

$20 - CPACKET\_START 7

$21 - CPACKET\_SKIP 7

$23 - CPACKET\_JMPMEM 7

$90 - CPACKET\_SECT 7

$91 - CPACKET\_DATA 7

$92 - CPACKET\_END 7

$93 - CPACKET\_CLEAR 7

$50 - CPACKET\_ALARM 7

$32 - CPACKET\_TIME 7

$70 - CPACKET\_MEM 7

$71 - CPACKET\_BEEPS 7

The Display 7

The TOP/MIDDLE Character Set 7

The Bottom Character set 7

Memory Map 7

Datalink Overview Memory Map 7

Differences between the 150 and 150S 7

Dealing with the Differences 7

Accessing the EEPROM 7

Sound Hardware 7

Hardware Tones 7

Important Terms: 7

The format of a sound scheme 7

Sound Files 7

Wristapp Programming Reference 7

The Processor 7

Tools 7

ASM6805 (2 months later) 7

The .ZSM file Format 7

System Routine Definitions 7

Program Layout Basics 7

Wristapp Interface Entries 7

Strings and Data 7

.ZAP File Format 7

Getting Started 7

The State Table 7

Special State Tables 7

Nested Apps 7

Button Events 7

Timer Events 7

Other Events 7

Event Constants 7

State Table Values 7

Timer Constants 7

Classes of Callable Functions 7

Anniversary support routines 7

Scanning support 7

Appointment support 7

Internal 7

Indiglo support 7

Sound Support 7

Event support 7

Packet/EEProm Support 7

INST Support 7

Scrolling Messages 7

Blinking routines 7

Update functions 7

Format Routines 7

Line routines 7

Installing a Wristapp 7

My Wristapps 7

Other People’s Wristapps 7

Plans for Wristapps 7

Wristapp Programming Tutorial 7

A First Wristapp - Hello World 7

Getting Input 7

Better Input - Update 7

Showing Selection - Blink routines 7

Entering Digits - PASSWD sample 7

Getting time and Input - DAYFIND sample 7

Playing With Sound - TestSnd example 7

Using Callbacks - Endoff example 7

Using 3 States - HexDump example 7

Dumping the EEPROM - promdump example 7

Tracking Money - Spend Watch example 7

Creating a Sound Scheme - Sound1 example 7

Random Numbers and Marquis - 3Ball example 7

Playing Hourly Chimes - Ships Bells example 7

# About the DataLink

## So, What is a Datalink?

The Datalink is a pretty neat watch that Timex created which allows you to download information just by pointing the watch at the display screen.  You have probably seen the commercials where the dog and cat play around with reprogramming the appointments on the watch.

What makes the watch interesting to me is that you can actually write programs for it.  Although Timex did not document how to do this, it turned out not to be too difficult to figure out how to write code for the watch.  Of course explaining how to do that is a bit more difficult, but that is what this document is all about.

## There are four basic models of the Datalink

1. The original 75 model which allowed you to download phone numbers, alarms, lists, anniversaries, and appointments.
2. The updated 150 model which doubled the download speed and increased the amount of memory for storing those phone numbers, alarms, lists, anniversaries, and appointments.  Timex also was kind enough to give us the ability to download wristApps to extend the functionality of the watch.
3. The smaller 150s model which is nearly identical to the 150 in capabilities.  This was introduced for the 1996 Christmas season as a lady’s watch.
4. The Ironman Datalink watch. Some people called this the 150r, but that is not the correct designation. While this watch is similar to the 150 and 150s with respect to capacity, it does not support downloading of wristapps.

### What programs can I load in the 150/150s?

Timex ships several useful Wristapps with the 150 in the box:

1. Note - Used for copying up to 255 characters of text (30-40 words) to the watch. It is useful for storing directions, etc. that need to be readily accessible.
2. Melody Tester - Used for testing Watch Tones on the watch. It sure beats waiting around for the appointment beep to go off.
3. Stopwatch - A chronograph that times events by starting from zero and counting up.
4. Adjustable Timer - Allows setting of a time to be counted from 1 minute to 100 hours, in 1 minute increments.
5. Preset Timer - The Preset Countdown Timer that allows for quick selection of the following preset times: 5, 10, 15 20, 30, 45, or 60 minutes.
6. Week of the Year (U.S.) - Displays what week of the year it is, what day of the year it is, and how many days are left in the year.
7. Week of the Year (International) - Displays what week of the year it is, what day of the year it is, and how many days are left in the year.

You can also purchase the optional Wristapps, which give you a few other useful wristapps:

1. Golf - A golfer’s electronic scorecard. Enter the number of strokes per hole and let the watch calculate the total for the round and the front and back nine. You can recall your totals or hole scores at any time.
2. CopyMe Game - A memory game. The watch displays a sequence of 0’s that you must duplicate using the watch’s buttons. If you are successful, the watch adds another step to the sequence. Make it through 15 steps and you win!
3. Pulse - Gives you a quick estimate of your pulse rate. Feel for your pulse. When the watch beeps, start counting beats. When you count ten, press a button, and the watch calculates your pulse. It’s a great workout companion.
4. World Time - Displays the time in each of the 24 time zones around the world.
5. Conversion - Gives you a table for converting values from one unit to another.

# Datalink Technical Details

## Download Protocol

### Synchronization Process

Before you can start sending any data to the DataLink, you have to send a series of sync bytes:

$55 (the watch has to see 4 in a row to be happy about it)

Once the watch has gotten the Sync bytes, it will look for a series of at least four $AA or $BF bytes to go into an initialization mode.

Once in initialization mode, it will start looking for the data bytes. If it sees a $EF, it will treat that as an escape byte and read in the next byte regardless of what it is (this allows the first byte of the packet to be a $55, $AA, $BF or even $EF).

Once it has gone into data transfer mode, it expects a series of 2 byte groups where the low bit of the first byte and the high bit of the second byte (I call these middle bits) must match to be sync bits. It expects these sync bits to alternate between 0 and 1. Any 2-byte group that does not match this will be thrown out. Also, if no valid bytes are received within 1/5 second, the transfer operation is aborted.

#### Sync Bits

With these sync bits, you can only transfer 14 bits of data for every 16 bits sent. (There are actually 2 extra sync bits on the screen to act as start and stop bits). If you look at it, that means that you can get 7 bytes transferred for every 8 bytes sent. The organization of these bits is:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | b | c | d | e | f | g | - |  | - | i | j | k | l | m | n | h |
| Q | r | s | t | u | o | p | + |  | + | z | y | A | B | v | w | x |
| G | H | I | C | D | E | F | - |  | - | O | P | J | K | L | M | N |
| W | Q | R | S | T | U | V | + |  | + | X | Y | Z | 1 | 2 | 3 | 4 |

Where - and + represent the sync bits (zero and one) in the byte pairs. If you decode these bits into the corresponding bytes, you get:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | b | c | d | e | f | g | h |
| I | j | k | l | m | n | o | p |
| Q | r | s | t | u | v | w | x |
| Y | z | A | B | C | D | E | F |
| G | H | I | J | K | L | M | N |
| O | P | Q | R | S | T | U | V |
| W | X | Y | Z | 1 | 2 | 3 | 4 |

Note that you always have to send in byte pairs, but the code is smart enough to throw away an extra byte which does not fit in a packet.   All packets end with a 2 byte 16-Bit CRC.

I think that the most interesting packet of all of this is the [CPACKET\_JMPMEM](#CPACKET_JMPMEM). It is possible to reset the watch by just sending this packet in the stream:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 09 | 23 | 04 | 3e | 18 | 94 | 81 | <crc-16> |

What this does is tells it to jump to location 04 3e which happens to be the address of where the 4th byte in the packet is stored. The code executes the 18 94 which is a BSET 4,TIMER\_FLAGS followed by an 81 = RTS. When the watch sees that 4,TIMER\_FLAGS has been set, it will run the watch through a complete reset cycle. There are a lot of other fun things that you can do. For example, you can play a tune during the download by storing new values at location 0335. So the packet:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0c | 23 | 04 | 3e | a6 | 01 | c7 | 03 | 35 | 81 | <crc-16> |

Would change the download tone to be a LOW C. Replace the 01 with any value up to 0f and you can actually play a tune as it is downloading. (The note at $0335 is played after each packet).

You can also use this code to indicate a status on the watch by setting the individual segments on the bottom:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0d | 23 | 04 | 3e | a6 | 48 | b7 | 1d | 19 | 1e | 81 | <crc-16> |

Would turn on the AM indicator. Of course since you can’t look at the watch while it is downloading, it would be little silly. However, this can be a great debug aid for someone working on the download protocol since the symbols are not cleared out once the download process starts.

The [CPACKET\_MEM](#CPACKET_MEM) packet is also pretty useful. You can use it to set any of the locations in ram to a particular value. This might be useful if you know that you have a certain Wristapp already loaded and you want to change some data stored in the wristapp. All you need is the address to store the data in and the data that you want to put there.

### Packet Format

#### $20 - CPACKET\_START

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $20 - CPACKET\_START |
| 2 | $00 |
| 3 | $00 |
| 4 | Version: 3=V2.0 for the 150, 4=V2.1 for the 150s |
| 5 | CRC-16 High |
| 6 | CRC-16 Low |

#### $21 - CPACKET\_SKIP

This skip packet does get sent to the Datalink, but its contents are completely ignored.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $21 – CPACKET\_SKIP |
| 2 | <ignored> |
| 3 | CRC-16 High |
| 4 | CRC-16 Low |

#### $23 - CPACKET\_JMPMEM

This JMPMEM packet is useful for jumping to/calling specific locations in memory during the download process.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $23 – CPACKET\_JMPMEM |
| 2 | Address High to jump to |
| 3 | Address low to jump to |
| 4 | CRC-16 High |
| 5 | CRC-16 Low |

#### $90 - CPACKET\_SECT

This is the Initialization packet to start loading a section. There are three formats based on the section to be loaded.

$90 - CPACKET\_SECT - Format 1

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $90 – CPACKET\_SECT |
| 2 | $01 - CLOAD\_EEPROM - Load up EEProm data |
| 3 | Number of CPACKET\_DATA packets to follow |
| 4 | CRC-16 High |
| 5 | CRC-16 Low |

$90 - CPACKET\_SECT - Format 2

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $90 – CPACKET\_SECT |
| 2 | $02 – CLOAD\_WRISTAPP - Load a new Wristapp |
| 3 | Number of CPACKET\_DATA packets to follow |
| 4 | Value to be stored in COMM\_010e |
| 5 | CRC-16 High |
| 6 | CRC-16 Low |

$90 - CPACKET\_SECT - Format 3

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $90 – CPACKET\_SECT |
| 2 | $03 - CLOAD\_SOUND - Load a new sound scheme |
| 3 | Number of CPACKET\_DATA packets to follow |
| 4 | Base offset for the sound (should be $100-length of the sound) |
| 5 | CRC-16 High |
| 6 | CRC-16 Low |

#### $91 - CPACKET\_DATA

This is the data packet sent after a CPACKET\_SECT. The number of packets sent will be dependent on the section and is indicated in the CPACKET\_SECT packet. Once these packets start getting sent, there should be no other packets until a CPACKET\_END is encountered (although there is really no error checking done on it). If the download is terminated without the last CPACKET\_END being seen or the right number of CPACKET\_DATA packets, the entire section is ignored.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $91 - CPACKET\_DATA |
| 2 | <ignored> (probably address high) |
| 3 | <ignored> (probably address low) |
| 4 .. n+4 | n Databytes to be stored |
| n+5 | CRC-16 High |
| n+6 | CRC-16 Low |

#### $92 - CPACKET\_END

This packet marks the end of a section.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $92 - CPACKET\_END |
| 2 | Section (1=CLOAD\_EEPROM, 2=CLOAD\_WRISTAPP, 3=CLOAD\_SOUND) |
| 3 | CRC-16 High |
| 4 | CRC-16 Low |

#### $93 - CPACKET\_CLEAR

This Packet is used to clear out a section.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $93 - CPACKET\_CLEAR |
| 2 | Section to clear (CLOAD\_EEPROM, CLOAD\_WRISTAPP, CLOAD\_SOUND) |
| 3 | CRC-16 High |
| 4 | CRC-16 Low |

#### $50 - CPACKET\_ALARM

This packet is used to set the alarm information for a single alarm.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $50 - CPACKET\_ALARM |
| 2 | Alarm Number (1-5) |
| 3 | Alarm Hour (0-23) |
| 4 | Alarm Minute (0-59) |
| 5 | <ignored> |
| 6 | <ignored> |
| 7 | Alarm String character 1 |
| 8 | Alarm String character 2 |
| 9 | Alarm String character 3 |
| 10 | Alarm String character 4 |
| 11 | Alarm String character 5 |
| 12 | Alarm String character 6 |
| 13 | Alarm String character 7 |
| 14 | Alarm String character 8 |
| 15 | Alarm enable 0=disable, non-zero=enable |
| 16 | CRC-16 High |
| 17 | CRC-16 Low |

#### $32 - CPACKET\_TIME

This single packet is used to set the time. It should be sent early in the process in ensure the best synchronization with the CPU clock time.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $32 – CPACKET\_TIME |
| 2 | Time zone selector (1=Time zone 1) |
| 3 | Seconds (0-59) |
| 4 | Hour (0-23) |
| 5 | Minute (0-59) |
| 6 | Month of the year (1-12) |
| 7 | Day of the month (1-31) |
| 8 | Current year (mod 1900) |
| 9 | Time Zone Name character 1 |
| 10 | Time Zone Name character 2 |
| 11 | Time Zone Name character 3 |
| 12 | Day of the week (0=Monday...6=Sunday) |
| 13 | 12/24 hour selector (1=12 Hour format, anything other than 1=24 hour format) |
| 14 | Time zone date format |
| 15 | CRC-16 High |
| 16 | CRC-16 Low |

#### $70 - CPACKET\_MEM

This packet is used to store a number of bytes into memory at a fixed location. Note that it is not used for loading up a wristapp because other information has to be reset when a wristapp has been loaded.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $70 – CPACKET\_MEM |
| 2 | High byte of memory address |
| 3 | Low byte of memory address |
| 4..n+3 | Data to be stored into memory |
| n+4 | CRC-16 High |
| n+5 | CRC-16 Low |

#### $71 - CPACKET\_BEEPS

This packet is used to control the hourly chimes and button beep flags.

|  |  |
| --- | --- |
| 0 | Packet Length |
| 1 | $71 - CPACKET\_BEEPS |
| 2 | Enable Hourly chimes flag (0=Disable, Non-Zero=Enable) |
| 3 | Enable Button beep flag (0=Disable, Non-Zero=Enable) |
| 4 | CRC-16 High |
| 5 | CRC-16 Low |

## The Display

The DataLink display has 4 basic areas when it comes to programming. For convenience, I call them simply:

1. TOP - The top 6 digits. Each of these digits are represented by 9 segments which can be individually controlled. There are dash and period separators between the second/third and the forth/fifth digits for displaying dates. There is also a dash separator between the third and forth digits which is used for telephone numbers. It also has a tic mark before the first digit as a shorthand for the first two digits of the year.
2. SYMBOLS - The AM/PM, Reminder, Night Mode, Alarm, and Note symbols. These tend to only be used by the Time app.
3. MIDDLE - Like the TOP area, the Middle area also consists of 6 digits each made up of 9 segments. For separators between the second and third digits, you can use a colon, period, or a dash. The fourth and fifth digits can be separated by a dash or a period.
4. BOTTOM - The bottom 8 digits which are each represented by a 5 by 5 matrix of pixels that can be individually addressed. The ROMs also support a series of scrolling routines to allow a message to be scrolled across the bottom at a nice even rate.

What is really nice about the watch is that every segment on the display is individually addressable. For convenience, we use a notation of BIT:OFFSET to indicate how to address the segment. What this means is that you need to set DISP\_ROW ($001d) to the OFFSET value and then set/clear the BIT in DISP\_COL ($001e) to turn on/off the corresponding segment. For example, if you wanted to turn on the AM indicator on the 150 which is referred to as 4:48, you would do:

LDA #$48

STA DISP\_ROW

BSET 4,DISP\_COL

Here’s a quick overview of the display. All of the segments are clickable so that you can determine the way to set/clear that segment. This is a Java applet, so if your browser is not capable of supporting Java, you won’t be able to see it. When you click on the segment, it will hi-light in red and display the appropriate set values on the status bar. Value1 will be what you use for the 150 and Value2 will be for the 150S.

### The TOP/MIDDLE Character Set

The [TOP](#TOP) and [MIDDLE](#MIDDLE) lines only allow for 32 different characters to be displayed (unless of course you do it all yourself). For convenience, we refer to this character set as the TIMEX6 character set. All of the Wristapps that are written use the TIMEX6 macro to convert ASCII strings to this set. Because you have to use the number zero for the letter O and the number five for the letter S, the TIMEX6 macro will handle the conversion for you. The characters that can’t easily be displayed are: J K Q V X Y. Fortunately, they aren’t used in a lot of words (except of course my first name :-).

The TIMEX6 character set does allow for the names of all the internal Apps to be displayed. It is important to be aware of this limited character set when choosing the name of your Wristapp, otherwise you won’t be able to display it easily when someone switches to the app.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$00** | **$01** | **$02** | **$03** | **$04** | **$05** | **$06** | **$07** | **$08** | **$09** | **$0a** | **$0b** | **$0c** | **$0d** | **$0e** | **$0f** |
| [0](#T6_0) | [1](#T6_1) | [2](#T6_2) | [3](#T6_3) | [4](#T6_4) | [5](#T6_5) | [6](#T6_6) | [7](#T6_7) | [8](#T6_8) | [9](#T6_9) | [A](#T6_A) | [B](#T6_B) | [C](#T6_C) | [D](#T6_D) | [E](#T6_E) | [F](#T6_F) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$10** | **$11** | **$12** | **$13** | **$14** | **$15** | **$16** | **$17** | **$18** | **$19** | **$1a** | **$1b** | **$1c** | **$1d** | **$1e** | **$1f** |
| [G](#T6_G) | [H](#T6_H) | [:](#T6_COLON) | [L](#T6_L) | [M](#T6_M) | [N](#T6_N) | [P](#T6_P) | [R](#T6_R) | [T](#T6_T) | [U](#T6_U) | [W](#T6_W) | [Y](#T6_Y) | [r](#T6_SR) | [\_](#T6_BLANK) | [-](#T6_MINUS) | [+](#T6_PLUS) |

The routines which are useful for putting strings on the top and middle lines are:

|  |  |
| --- | --- |
| [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP) | ??? |
| [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID) | ??? |
| [PUTMSG1](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSG1) | ??? |
| [PUTMSG2](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSG2) | ??? |
| [CLEARTOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARTOP) | ??? |
| [CLEARMID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARMID) | ??? |
| [CLEARTOP12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARTOP12) | ??? |
| [CLEARTOP34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARTOP34) | ??? |
| [CLEARTOP56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARTOP56) | ??? |
| [CLEARMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARMID12) | ??? |
| [CLEARMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARMID34) | ??? |
| [CLEARMID56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARMID56) | ??? |
| [PUTLINE1](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTLINE1) | ??? |
| [PUTLINE2](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTLINE2) | ??? |
| [PUTTOP12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP12) | ??? |
| [PUTTOP34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP34) | ??? |
| [PUTTOP56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP56) | ??? |
| [PUTMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID12) | ??? |
| [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34) | ??? |
| [PUTMID56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID56) | ??? |

You can see what all of these are displayed as below.

|  |  |  |  |
| --- | --- | --- | --- |
| **$00 – 0** | **$01 - 1** | **$02 - 2** | **$03 - 3** |
| |=====|  | |  | |  | |  | |  | |  |=====| | |  |  |  |  |  |  | | =====|  |  |  =====|  |  |  |===== | =====|  |  |  =====|  |  |  =====| |

|  |  |  |  |
| --- | --- | --- | --- |
| **$04 - 4** | **$05 - 5** | **$06 - 6** | **$07 - 7** |
| | |  | |  | |  |=====|  |  |  | | |=====  |  |  |=====  |  |  =====| | |=====  |  |  |=====  | |  | |  |=====| | =====|  |  |  |  |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **$08 - 8** | **$09 - 9** | **$0a - A** | **$0b - B** |
| |=====|  | |  | |  |=====|  | |  | |  |=====| | |=====|  | |  | |  |=====|  |  |  =====| | |=====|  | |  | |  |=====|  | |  | |  | | | =====|  | |  | |  =====|  | |  | |  =====| |

|  |  |  |  |
| --- | --- | --- | --- |
| **$0c - C** | **$0d - D** | **$0e - E** | **$0f - F** |
| |=====  |  |  |  |  |  |===== | =====|  | |  | |  |  | |  | |  =====| | |=====  |  |  |=====  |  |  |===== | |=====  |  |  |=====  |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **$10 - G** | **$11 - H** | **$12 - I** | **$13 - L** |
| |=====  |  |  |  | |  | |  |=====| | | |  | |  | |  |=====|  | |  | |  | | | |  |    |  | | |  |  |  |  |  |  |===== |

|  |  |  |  |
| --- | --- | --- | --- |
| **$14 - M** | **$15 - N** | **$16 - P** | **$17 - R** |
| |=====|  | | |  | | |  | |  | |  | |  | | | |=====|  | |  | |  | |  | |  | |  | | | |=====|  | |  | |  |=====|  |  |  | | |=====|  | |  | |  |=====|  | |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **$18 - T** | **$19 - U** | **$1a - W** | **$1b - Y** |
| =====  |  |    |  | | | |  | |  | |  | |  | |  | |  |=====| | | |  | |  | |  | |  | | |  | | |  |=====| | | |  | |  | |  |=====|  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| **$1c - r** | **$1d -** | **$1e - -** | **$1f - +** |
| =====  |  |  | |  | ===== | |  |  =====  |  | |

### The Bottom Character set

The [BOTTOM](#BOTTOM) line has a slightly richer character set which we call the TIMEX character set. It allows for 64 different characters, includes the entire upper case alphabet and quite a few special symbols. All of these characters are drawn on a 5x5 dot matrix.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$00** | **$01** | **$02** | **$03** | **$04** | **$05** | **$06** | **$07** | **$08** | **$09** | **$0a** | **$0b** | **$0c** | **$0d** | **$0e** | **$0f** |
| [0](#T8_0) | [1](#T8_1) | [2](#T8_2) | [3](#T8_3) | [4](#T8_4) | [5](#T8_5) | [6](#T8_6) | [7](#T8_7) | [8](#T8_8) | [9](#T8_9) | [A](#T8_A) | [B](#T8_B) | [C](#T8_C) | [D](#T8_D) | [E](#T8_E) | [F](#T8_F) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$10** | **$11** | **$12** | **$13** | **$14** | **$15** | **$16** | **$17** | **$18** | **$19** | **$1a** | **$1b** | **$1c** | **$1d** | **$1e** | **$1f** |
| [G](#T8_G) | [H](#T8_H) | [I](#T8_I) | [J](#T8_J) | [K](#T8_K) | [L](#T8_L) | [M](#T8_M) | [N](#T8_N) | [O](#T8_O) | [P](#T8_P) | [Q](#T8_Q) | [R](#T8_R) | [S](#T8_S) | [T](#T8_T) | [U](#T8_U) | [V](#T8_V) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$20** | **$21** | **$22** | **$23** | **$24** | **$25** | **$26** | **$27** | **$28** | **$29** | **$2a** | **$2b** | **$2c** | **$2d** | **$2e** | **$2f** |
| [W](#T8_W) | [X](#T8_X) | [Y](#T8_Y) | [Z](#T8_Z) | [\_](#T8_BLANK) | [!](#T8_EXCLAIM) | [“](#T8_DQUOTE) | [#](#T8_POUND) | [>](#T8_DOLLAR) | [%](#T8_PCT) | [&](#T8_AMPER) | [‘](#T8_QUOTE) | [(](#T8_LPAREN) | [)](#T8_RPAREN) | [\*](#T8_STAR) | [+](#T8_PLUS) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **$30** | **$31** | **$32** | **$33** | **$34** | **$35** | **$36** | **$37** | **$38** | **$39** | **$3a** | **$3b** | **$3c** | **$3d** | **$3e** | **$3f** |
| [,](#T8_COMMA) | [-](#T8_DASH) | [.](#T8_DOT) | [/](#T8_SLASH) | [:](#T8_COLON) | [\](#T8_BSLASH) | [DIV](#T8_DIVIDE) | [=](#T8_EQUAL) | [BELL](#T8_BELL) | [?](#T8_QMARK) | [\_](#T8_ULINE) | [CHK](#T8_CHECK) | [PREV](#T8_PREV) | [NEXT](#T8_NEXT) | [BLOCK](#T8_BLOCK) | [SEP](#T8_SEP) |

The routines which are useful for putting strings on the top and middle lines are:

|  |  |
| --- | --- |
| [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8) | ??? |
| [PUTMSGXBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGXBOT) | ??? |
| [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT) | ??? |
| [PUTBOT678](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTBOT678) | ??? |
| [PUTLINE3](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTLINE3) | ??? |
| [PUT\_LETTERX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT_LETTERX) | ??? |
| [PUTSCROLLMSG](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTSCROLLMSG) | ??? |
| [SCROLLMSG](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#SCROLLMSG) | ??? |
| [SCROLLMSG\_CONT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#SCROLLMSG_CONT) | ??? |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$00-0** | **$01-1** | **$02-2** | **$03-3** | **$04-4** | **$05-5** | **$06-6** | **$07-7** |
| \_\_@@\_  \_@\_\_@  \_@\_\_@  \_@\_\_@  \_\_@@\_ | \_\_@\_\_  \_@@\_\_  \_\_@\_\_  \_\_@\_\_  \_\_@\_\_ | @@@@\_  \_\_\_\_@  @@@  @\_\_\_\_  @@@@@ | @@@@\_  \_\_\_\_@  @@@  \_\_\_\_@  @@@@\_ | @\_\_@\_  @\_\_@\_  @@@@@  \_\_\_@\_  \_\_\_@\_ | @@@@@  @\_\_\_\_  @@@@\_  \_\_\_\_@  @@@@\_ | \_@@@@  @\_\_\_\_  @@@@\_  @\_\_\_@  @@@ | @@@@@  \_\_\_\_@  \_\_\_@\_  \_\_@\_\_  \_\_@\_\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$08-8** | **$09-9** | **$0a-A** | **$0b-B** | **$0c-C** | **$0d-D** | **$0e-E** | **$0f-F** |
| \_@@@\_  @\_\_\_@  \_@@@\_  @\_\_\_@  \_@@@\_ | \_@@@\_  @\_\_\_@  \_@@@@  \_\_\_\_@  @@@@\_ | \_@@@\_  @\_\_\_@  @@@@@  @\_\_\_@  @\_\_\_@ | @@@@\_  @\_\_\_@  @@@@\_  @\_\_\_@  @@@@\_ | \_@@@@  @\_\_\_\_  @\_\_\_\_  @\_\_\_\_  \_@@@@ | @@@@\_  @\_\_\_@  @\_\_\_@  @\_\_\_@  @@@@\_ | @@@@@  @\_\_\_\_  @@@@\_  @\_\_\_\_  @@@@@ | @@@@@  @\_\_\_\_  @@@@\_  @\_\_\_\_  @\_\_\_\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$10-G** | **$11-H** | **$12-I** | **$13-J** | **$14-K** | **$15-L** | **$16-M** | **$17-N** |
| \_@@@@  @\_\_\_\_  @\_@@@  @\_\_\_@  \_@@@@ | @\_\_\_@  @\_\_\_@  @@@@@  @\_\_\_@  @\_\_\_@ | @@@  \_\_@\_\_  \_\_@\_\_  \_\_@\_\_  @@@ | \_\_@@@  \_\_\_@\_  \_\_\_@\_  @\_\_@\_  \_@@\_\_ | @\_\_\_@  @\_\_@\_  @\_@\_\_  @@\_@\_  @\_\_\_@ | @\_\_\_\_  @\_\_\_\_  @\_\_\_\_  @\_\_\_\_  @@@@@ | @\_\_\_@  @@\_@@  @\_@\_@  @\_@\_@  @\_\_\_@ | @\_\_\_@  @@\_\_@  @\_@\_@  @\_\_@@  @\_\_\_@ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$18-O** | **$19-P** | **$1a-Q** | **$1b-R** | **$1c-S** | **$1d-T** | **$1e-U** | **$1f-V** |
| @@@  @\_\_\_@  @\_\_\_@  @\_\_\_@  @@@ | @@@@\_  @\_\_\_@  @@@@\_  @\_\_\_\_  @\_\_\_\_ | @@@  @\_\_\_@  @\_@\_@  @\_\_@\_  \_@@\_@ | @@@@\_  @\_\_\_@  @@@@\_  @\_\_@\_  @\_\_\_@ | \_@@@@  @\_\_\_\_  @@@  \_\_\_\_@  @@@@\_ | @@@@@  \_\_@\_\_  \_\_@\_\_  \_\_@\_\_  \_\_@\_\_ | @\_\_\_@  @\_\_\_@  @\_\_\_@  @\_\_\_@  @@@ | @\_\_\_@  @\_\_\_@  @\_\_\_@  \_@\_@\_  \_\_@\_\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$20-W** | **$21-X** | **$22-Y** | **$23-Z** | **$24-** | **$25-!** | **$26-“** | **$27-#** |
| @\_\_\_@  @\_\_\_@  @\_@\_@  @@@  \_@\_@\_ | @\_\_\_@  \_@\_@\_  \_\_@\_\_  \_@\_@\_  @\_\_\_@ | @\_\_\_@  \_@\_@\_  \_\_@\_\_  \_\_@\_\_  \_\_@\_\_ | @@@@@  \_\_\_@\_  \_\_@\_\_  \_@\_\_\_  @@@@@ | \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_ | \_\_@\_\_  \_\_@\_\_  \_\_@\_\_  \_\_\_\_\_  \_\_@\_\_ | \_@\_@\_  \_@\_@\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_ | \_@\_@\_  @@@@@  \_@\_@\_  @@@@@  \_@\_@\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$28-$** | **$29-%** | **$2a-&** | **$2b-‘** | **$2c-(** | **$2d-)** | **$2e-\*** | **$2f-+** |
| \_@@@@  @\_@\_\_  @@@  \_\_@\_@  @@@@\_ | @@\_\_@  \_\_\_@\_  \_\_@\_\_  \_@\_\_\_  @\_\_@@ | \_@\_\_\_  @\_@\_\_  \_@@\_@  @\_\_@\_  \_@@\_@ | \_\_@\_\_  \_@\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_ | \_\_@\_\_  \_@\_\_\_  \_@\_\_\_  \_@\_\_\_  \_\_@\_\_ | \_\_@\_\_  \_\_\_@\_  \_\_\_@\_  \_\_\_@\_  \_\_@\_\_ | @\_@\_@  @@@  @@@@@  @@@  @\_@\_@ | \_\_@\_\_  \_\_@\_\_  @@@@@  \_\_@\_\_  \_\_@\_\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$30-,** | **$31--** | **$32-.** | **$33-/** | **$34-:** | **$35-\** | **$36-** | **$37-=** |
| \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_@\_\_  \_@\_\_\_ | \_\_\_\_\_  \_\_\_\_\_  @@@@@  \_\_\_\_\_  \_\_\_\_\_ | \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_@\_\_ | \_\_\_\_@  \_\_\_@\_  \_\_@\_\_  \_@\_\_\_  @\_\_\_\_ | \_\_\_\_\_  \_\_@\_\_  \_\_\_\_\_  \_\_@\_\_  \_\_\_\_\_ | @\_\_\_\_  \_@\_\_\_  \_\_@\_\_  \_\_\_@\_  \_\_\_\_@ | \_\_@\_\_  \_\_\_\_\_  @@@@@  \_\_\_\_\_  \_\_@\_\_ | \_\_\_\_\_  @@@@@  \_\_\_\_\_  @@@@@  \_\_\_\_\_ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **$38->** | **$39-?** | **$3a-\_** | **$3b-** | **$3c-** | **$3d-** | **$3e-** | **$3f-** |
| \_\_@\_\_  \_@@@\_  \_@@@\_  @@@@@  \_\_@\_\_ | \_@@\_\_  @\_\_@\_  \_\_@\_\_  \_\_\_\_\_  \_\_@\_\_ | \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  \_\_\_\_\_  @@@@@ | \_\_\_\_@  \_\_\_@\_  @\_@\_\_  \_@\_\_\_  @@@@@ | \_\_\_\_@  \_\_@@@  @@@@@  \_\_@@@  \_\_\_\_@ | @\_\_\_\_  @@@\_\_  @@@@@  @@@\_\_  @\_\_\_\_ | @@@@@  @@@@@  @@@@@  @@@@@  @@@@@ | \_\_\_\_\_  \_@@@\_  \_@@@\_  \_@@@\_  \_\_\_\_\_ |

|  |  |  |
| --- | --- | --- |
| Alarm | 4:1C | 4:1A |
| AM | 4:48 | 4:46 |
| M1A | 4:42 | 4:40 |
| M1B | 3:40 | 3:3E |
| M1C | 2:40 | 2:3E |
| M1D | 2:46 | 2:44 |
| M1E | 3:46 | 3:44 |
| M1F | 4:46 | 4:44 |
| M1G | 3:44 | 3:42 |
| M1H | 4:44 | 4:42 |
| M1I | 2:44 | 2:42 |
| M2A | 4:3A | 4:38 |
| M2B | 3:38 | 3:36 |
| M2C | 2:38 | 2:36 |
| M2D | 2:3E | 2:3C |
| M2E | 3:3E | 3:3C |
| M2F | 4:3E | 4:3C |
| M2G | 3:3C | 3:3A |
| M2H | 4:3C | 4:3A |
| M2I | 2:3C | 2:3A |
| M3A | 4:30 | 4:2E |
| M3B | 3:2E | 3:2C |
| M3C | 2:2E | 2:2C |
| M3D | 2:34 | 2:32 |
| M3E | 3:34 | 3:32 |
| M3F | 4:34 | 4:32 |
| M3G | 3:32 | 3:30 |
| M3H | 4:32 | 4:30 |
| M3I | 2:32 | 2:30 |
| M4A | 4:28 | 4:26 |
| M4B | 3:26 | 3:24 |
| M4C | 2:26 | 2:24 |
| M4D | 2:2C | 2:2A |
| M4E | 3:2C | 3:2A |
| M4F | 4:2C | 4:2A |
| M4G | 3:2A | 3:28 |
| M4H | 4:2A | 4:28 |
| M4I | 2:2A | 2:28 |
| M5A | 4:1E | 4:1C |
| M5B | 3:1C | 3:1A |
| M5C | 2:1C | 2:1A |
| M5D | 2:22 | 2:20 |
| M5E | 3:22 | 3:20 |
| M5F | 4:22 | 4:20 |
| M5G | 3:20 | 3:1E |
| M5H | 4:20 | 4:1E |
| M5I | 2:20 | 2:1E |
| M6A | 4:10 | 4:0E |
| M6B | 3:0E | 3:0C |
| M6C | 2:0E | 2:0C |
| M6D | 2:14 | 2:12 |
| M6E | 3:14 | 3:12 |
| M6F | 4:14 | 4:12 |
| M6G | 3:12 | 3:10 |
| M6H | 4:12 | 4:10 |
| M6I | 2:12 | 2:10 |
| MC23 | 3:36 | 3:34 |
| MD45 | 3:24 | 3:22 |
| MP23 | 2:36 | 2:34 |
| MP45 | 2:24 | 2:22 |
| Night | 4:26 | 4:24 |
| Note | 4:0e | 4:0C |
| PM | 4:40 | 4:3E |
| Remind | 4:38 | 4:36 |
| S1A1 | 2:47 |  |
| S1A2 | 2:45 |  |
| S1A3 | 2:43 |  |
| S1A4 | 2:41 |  |
| S1A5 | 2:3F |  |
| S1B1 | 3:47 |  |
| S1B2 | 3:45 |  |
| S1B3 | 3:43 |  |
| S1B4 | 3:41 |  |
| S1B5 | 3:3F |  |
| S1C1 | 4:47 |  |
| S1C2 | 4:45 |  |
| S1C3 | 4:43 |  |
| S1C4 | 4:41 |  |
| S1C5 | 4:3F |  |
| S1D1 | 0:47 |  |
| S1D2 | 0:45 |  |
| S1D3 | 0:43 |  |
| S1D4 | 0:41 |  |
| S1D5 | 0:3F |  |
| S1E1 | 1:47 |  |
| S1E2 | 1:45 |  |
| S1E3 | 1:43 |  |
| S1E4 | 1:41 |  |
| S1E5 | 1:3F |  |
| S2A1 | 2:3D |  |
| S2A2 | 2:3B |  |
| S2A3 | 2:39 |  |
| S2A4 | 2:37 |  |
| S2A5 | 2:35 |  |
| S2B1 | 3:3D |  |
| S2B2 | 3:3B |  |
| S2B3 | 3:39 |  |
| S2B4 | 3:37 |  |
| S2B5 | 3:35 |  |
| S2C1 | 4:3D |  |
| S2C2 | 4:3B |  |
| S2C3 | 4:39 |  |
| S2C4 | 4:37 |  |
| S2C5 | 4:35 |  |
| S2D1 | 0:3D |  |
| S2D2 | 0:3B |  |
| S2D3 | 0:39 |  |
| S2D4 | 0:37 |  |
| S2D5 | 0:35 |  |
| S2E1 | 1:3D |  |
| S2E2 | 1:3B |  |
| S2E3 | 1:39 |  |
| S2E4 | 1:37 |  |
| S2E5 | 1:35 |  |
| S3A1 | 2:33 |  |
| S3A2 | 2:31 |  |
| S3A3 | 2:2F |  |
| S3A4 | 2:2D |  |
| S3A5 | 2:2B |  |
| S3B1 | 3:33 |  |
| S3B2 | 3:31 |  |
| S3B3 | 3:2F |  |
| S3B4 | 3:2D |  |
| S3B5 | 3:2B |  |
| S3C1 | 4:33 |  |
| S3C2 | 4:31 |  |
| S3C3 | 4:2F |  |
| S3C4 | 4:2D |  |
| S3C5 | 4:2B |  |
| S3D1 | 0:33 |  |
| S3D2 | 0:31 |  |
| S3D3 | 0:2F |  |
| S3D4 | 0:2D |  |
| S3D5 | 0:2B |  |
| S3E1 | 1:33 |  |
| S3E2 | 1:31 |  |
| S3E3 | 1:2F |  |
| S3E4 | 1:2D |  |
| S3E5 | 1:2B |  |
| S4A1 | 2:27 |  |
| S4A2 | 2:25 |  |
| S4A3 | 2:23 |  |
| S4A4 | 2:21 |  |
| S4A5 | 2:1F |  |
| S4B1 | 3:27 |  |
| S4B2 | 3:25 |  |
| S4B3 | 3:23 |  |
| S4B4 | 3:21 |  |
| S4B5 | 3:1F |  |
| S4C1 | 4:27 |  |
| S4C2 | 4:25 |  |
| S4C3 | 4:23 |  |
| S4C4 | 4:21 |  |
| S4C5 | 4:1F |  |
| S4D1 | 0:27 |  |
| S4D2 | 0:25 |  |
| S4D3 | 0:23 |  |
| S4D4 | 0:21 |  |
| S4D5 | 0:1F |  |
| S4E1 | 1:27 |  |
| S4E2 | 1:25 |  |
| S4E3 | 1:23 |  |
| S4E4 | 1:21 |  |
| S4E5 | 1:1F |  |
| S5A1 | 2:1D |  |
| S5A2 | 2:1B |  |
| S5A3 | 2:19 |  |
| S5A4 | 2:17 |  |
| S5A5 | 2:15 |  |
| S5B1 | 3:1D |  |
| S5B2 | 3:1B |  |
| S5B3 | 3:19 |  |
| S5B4 | 3:17 |  |
| S5B5 | 3:15 |  |
| S5C1 | 4:1D |  |
| S5C2 | 4:1B |  |
| S5C3 | 4:19 |  |
| S5C4 | 4:17 |  |
| S5C5 | 4:15 |  |
| S5D1 | 0:1D |  |
| S5D2 | 0:1B |  |
| S5D3 | 0:19 |  |
| S5D4 | 0:17 |  |
| S5D5 | 0:15 |  |
| S5E1 | 1:1D |  |
| S5E2 | 1:1B |  |
| S5E3 | 1:19 |  |
| S5E4 | 1:17 |  |
| S5E5 | 1:15 |  |
| S6A1 | 2:13 |  |
| S6A2 | 2:11 |  |
| S6A3 | 2:0F |  |
| S6A4 | 2:0D |  |
| S6A5 | 2:0B |  |
| S6B1 | 3:13 |  |
| S6B2 | 3:11 |  |
| S6B3 | 3:0F |  |
| S6B4 | 3:0D |  |
| S6B5 | 3:0B |  |
| S6C1 | 4:13 |  |
| S6C2 | 4:11 |  |
| S6C3 | 4:0F |  |
| S6C4 | 4:0D |  |
| S6C5 | 4:0B |  |
| S6D1 | 0:13 |  |
| S6D2 | 0:11 |  |
| S6D3 | 0:0F |  |
| S6D4 | 0:0D |  |
| S6D5 | 0:0B |  |
| S6E1 | 1:13 |  |
| S6E2 | 1:11 |  |
| S6E3 | 1:0F |  |
| S6E4 | 1:0D |  |
| S6E5 | 1:0B |  |
| S7A1 | 2:09 |  |
| S7A2 | 2:07 |  |
| S7A3 | 2:05 |  |
| S7A4 | 2:03 |  |
| S7A5 | 2:01 |  |
| S7B1 | 3:09 |  |
| S7B2 | 3:07 |  |
| S7B3 | 3:05 |  |
| S7B4 | 3:03 |  |
| S7B5 | 3:01 |  |
| S7C1 | 4:09 |  |
| S7C2 | 4:07 |  |
| S7C3 | 4:05 |  |
| S7C4 | 4:03 |  |
| S7C5 | 4:01 |  |
| S7D1 | 0:09 |  |
| S7D2 | 0:07 |  |
| S7D3 | 0:05 |  |
| S7D4 | 0:03 |  |
| S7D5 | 0:01 |  |
| S7E1 | 1:09 |  |
| S7E2 | 1:07 |  |
| S7E3 | 1:05 |  |
| S7E4 | 1:03 |  |
| S7E5 | 1:01 |  |
| S8A1 | 2:02 |  |
| S8A2 | 2:04 |  |
| S8A3 | 2:06 |  |
| S8A4 | 2:08 |  |
| S8A5 | 2:0a |  |
| S8B1 | 3:02 |  |
| S8B2 | 3:04 |  |
| S8B3 | 3:06 |  |
| S8B4 | 3:08 |  |
| S8B5 | 3:0a |  |
| S8C1 | 4:02 |  |
| S8C2 | 4:04 |  |
| S8C3 | 4:06 |  |
| S8C4 | 4:08 |  |
| S8C5 | 4:0a |  |
| S8D1 | 1:02 |  |
| S8D2 | 1:04 |  |
| S8D3 | 1:06 |  |
| S8D4 | 1:08 |  |
| S8D5 | 1:0a |  |
| S8E1 | 0:02 |  |
| S8E2 | 0:04 |  |
| S8E3 | 0:06 |  |
| S8E4 | 0:08 |  |
| S8E5 | 0:0a |  |
| T1A | 2:42 | 2:40 |
| T1B | 1:40 | 1:3E |
| T1C | 0:40 | 0:3E |
| T1D | 0:42 | 0:40 |
| T1E | 0:46 | 0:44 |
| T1F | 1:46 | 1:44 |
| T1G | 1:42 | 1:40 |
| T1H | 1:44 | 1:42 |
| T1I | 0:44 | 0:42 |
| T2A | 2:3A | 2:38 |
| T2B | 1:38 | 1:36 |
| T2C | 0:38 | 0:36 |
| T2D | 0:3A | 0:38 |
| T2E | 0:3E | 0:3C |
| T2F | 1:3E | 1:3C |
| T2G | 1:3A | 1:38 |
| T2H | 1:3C | 1:3A |
| T2I | 0:3C | 0:3A |
| T3A | 2:30 | 2:2E |
| T3B | 1:2E | 1:2C |
| T3C | 0:2E | 0:2C |
| T3D | 0:30 | 0:2E |
| T3E | 0:34 | 0:32 |
| T3F | 1:34 | 1:32 |
| T3G | 1:30 | 1:2E |
| T3H | 1:32 | 1:30 |
| T3I | 0:32 | 0:30 |
| T4A | 2:28 | 2:26 |
| T4B | 1:26 | 1:24 |
| T4C | 0:26 | 0:24 |
| T4D | 0:28 | 0:26 |
| T4E | 0:2C | 0:2A |
| T4F | 1:2C | 1:2A |
| T4G | 1:28 | 1:26 |
| T4H | 1:2A | 1:28 |
| T4I | 0:2A | 0:28 |
| T5A | 2:1E | 2:1C |
| T5B | 1:1c | 1:1A |
| T5C | 0:1c | 0:1A |
| T5D | 0:1e | 0:1C |
| T5E | 0:22 | 0:20 |
| T5F | 1:22 | 1:20 |
| T5G | 1:1E | 1:1C |
| T5H | 1:20 | 1:1E |
| T5I | 0:20 | 0:1E |
| T6A | 2:10 | 2:0E |
| T6B | 1:0e | 1:0C |
| T6C | 0:0e | 0:0C |
| T6D | 0:10 | 0:0E |
| T6E | 0:14 | 0:12 |
| T6F | 1:14 | 1:12 |
| T6G | 1:10 | 1:0E |
| T6H | 1:12 | 1:10 |
| T6I | 0:12 | 0:10 |
| TD23 | 1:36 | 1:34 |
| TD34 | 4:2E | 4:2C |
| TD45 | 1:24 | 1:22 |
| TP23 | 0:36 | 0:34 |
| TP45 | 0:24 | 0:22 |

## Memory Map

The Datalink is controlled by a custom 6805 which has 16K of ROM, 1.25K of Ram and 2.0K of EEProm.  Because the 6805 has a 15 bit address bus, all accesses wrap at 0800 to 0000 and repeat once again.  The EEProm is a serial device and does not appear in the accessible address space for the 6805.

### Datalink Overview Memory Map

|  |  |
| --- | --- |
| 0000-002A | 6805 Hardware registers |
| 002B-004F | Unused ram (probably not even mapped) |
| 0050-005F | System App local variables |
| 0060-0067 | Wristapp local variables |
| 0068-00C2 | System local variables |
| 00C3-00FF | Call stack |
| 0100-010F | EEProm control variables |
| 0110-0335 | Wristapp memory |
| 0336-0435 | Sound memory (starts high, low end can be used for a larger wristapp) |
| 0436-04FF | System upper ram |
| 0500-3FFF | Unused - This is a hole in the address space |
| 0400-7FFF | System ROM |

<More memory map stuff to come>

## Differences between the 150 and 150S

For Christmas 1996, Timex introduced a smaller version of the 150 called the 150s.  This watch has substantially the same hardware and capabilities as the 150, but in a smaller package.  You can tell the difference between the two by entering COMM Mode.  If the version on the bottom line is V2.0, then the watch is a 150.  If it says V2.1, it is a 150S.  There is also a newer release of the Datalink software (V2.1) for the 150s which works with all of the Datalink watches.  The older V2.0 software will not talk to the 150s.

It is not possible to run the same wristapp on both watches because of a few differences:

1. The addresses for the display segments have changed. Mostly this has been a simple subtraction of 2 from the offsets for addressing the display segments, but it also involved the shuffling of a couple of the pixels in the segments on the bottom line. Since turning on a segment is a hard coded constant, an application has to be recompiled for the different display.
2. To accommodate the change in display segments, a couple of ROM routines have been changed.  This resulted in a shuffling of the addresses of a number of routines within the watch.
3. To further complicate things, the order of a few routines in the ROM has been changed.  While the routines are exactly the same in both the 150 and the 150s, the location of these routines is never the same.
4. The [CPACKET\_START](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\Download.html#CPACKET_START) packet has a 4 for the version code instead of a 3.
5. Even with all this shuffling, the memory map for the low ram appears to be exactly identical, as does the actual 6805 hardware.

### Dealing with the Differences

Because the two watches are so different, you have to essentially write the same program twice with different targets for all of the system routines and any segment poking that is done.  The V2.1 software handles this by storing both copies of the code in the .ZAP file with a description field to identify which watch the software is targeted to.  When you identify the type of the watch to the DataLink software, it automatically chooses the right software to send.

## Accessing the EEPROM

The 2K EEProm in the Datalink is accessed over a serial interface and is not directly mapped to the 6805 address space.  The entries for all of the apps are stored sequentially in the EEProm with a length/flag byte at the front of each one.  When an entry is deleted, it is done by simply setting the high bit on the flag byte.  All of the internal software simply skips over the entry.  There is no code in the watch for shuffling the data in the EEProm.

When any data is downloaded to the EEProm, it essentially clears the EEProm pointers and starts again. This has the effect of deleting all Phone, List, Anniversary, and Appointment entries if you just load a single entry down to the watch.  However, the actual data in the EEProm is never cleared out except when new data overwrites.  This means that it is possible to dump out the data in the EEProm even if the watch has been reset or only one or two entries downloaded to it.

There are some internal routines for getting to the EEProm (to be documented later) and it is possible with some work to write code that allows you to store entries in the EEProm, but you would have to figure out how to shuffle the entries in the EEProm if you wanted to add an entry without deleting everything (this isn’t really as difficult as it sounds).

## Sound Hardware

The Datalink is capable of playing 14 tones by poking one of the following values into PORT\_SOUND (location $0028).  From experimentation, it appears that only the low nibble of whatever value is poked into this location is actually used.

It is my current working theory that there is a timer routine in the Datalink which is actually causing the resulting frequencies and it might be possible to generate other sounds by going through a slightly different mechanism to poke the sound hardware.

Note that if you use the built-in sound routines for playing sounds, you will find that the interrupt routines will happily readjust the hardware tones behind your back.

### Hardware Tones

|  |  |
| --- | --- |
| 0 | Tone\_END - This seems to generate silence |
| 1 | Low C |
| 2 | High C |
| 3 | Middle C |
| 4 | Very High C |
| 5 | High F (Reported to be a little bit lower than F) |
| 6 | Middle F |
| 7 | Low F |
| 8 | Very High G# (G-Sharp) |
| 9 | High G# (G-Sharp) |
| 10 | Middle G# (G-Sharp) |
| 11 | Low G# (G-Sharp) |
| 12 | High D |
| 13 | Middle D |
| 14 | Low D |
| 15 | Silence |

### Important Terms:

Sound Scheme - A set of sounds (this is the .SPC file in the SND directory of the Datalink application) which are downloaded to the watch. A sound scheme contains all the Soundlets and [Sound Sequences](#Sound_Sequence) for all 10 defined system sound values. This file is loaded in the watch so that the end of it is at $0435 in memory.

System Sound - is one of the 10 defined system sound values:

|  |  |  |
| --- | --- | --- |
| **Value** | **Symbol** | **Purpose** |
| $80 | SND\_NONE | No sound at all |
| $c1 | SND\_BUTTON | Button Beep |
| $c2 | SND\_RETURN | Return to time |
| $83 | SND\_HOURLY | Hourly Chime |
| $c4 | SND\_CONF | Confirmation |
| $85 | SND\_APPT | Appointment Beep |
| $86 | SND\_ALARM | Alarm Beep |
| $87 | SND\_DLOAD | Program Download |
| $88 | SND\_EXTRA | Extra sound |
| $89 | SND\_COMERR | Comm Error |
| $8a | SND\_DONE | Comm done |

Sound Sequence - The sequence of [soundlets](#soundlet) which are played to for a given [System Sound](#System_Sound). There can be as few as 1 Sound Sequence and as many as 10 different Sound Sequences. Each System Sound maps to one Sound Sequence although the same Sound Sequence can be used for more than one System Sound. A Sound Sequence is represented by two series of numbers.

The first series is called the Soundlet Count Table and consists of a series of one or more bytes where the last byte in the series has the high bit set ($80). For each entry in the Soundlet Count Table, the number of times that a sound is played is determined by clearing the tip bit and then using the resulting number as a count. So $81 indicates the last entry with a repeat count of 1. $A0 indicates the last entry with a repeat count of 20. $0A indicates an entry (with at least one more following it) with a repeat count of 10.

The second series is the Soundlet Pointer Table which consists of exactly the same number of entries as the Soundlet Count Table. Each entry in this table is simply a pointer to the start of the corresponding Soundlet

Soundlet - A sequence of [Notes](#Note) terminated by a 0 note. There is no practical limit on the number of notes in a Soundlet except for the total size of 256 bytes for the entire [Sound Scheme](#Sound_Scheme).

Note - A single sound to be played. The note consists of a single byte broken into two Nibbles. The high order nibble is the [tone](#tone) to be played and the low order nibble is the duration for that tone in 1/10th of a second intervals.

Tone - One of 14 tones supported by the sound hardware on the watch as well as the two values which produce silence:

## Sound Scheme Format

Given the default sounds in the ROM, I propose that this is how we would interpret and code them:

TONE\_END EQU $00 ; END

TONE\_LOW\_C EQU $10 ; Low C

TONE\_HI\_C EQU $20 ; High C

TONE\_MID\_C EQU $30 ; Middle C

TONE\_VHI\_C EQU $40 ; Very high C

TONE\_HI\_F EQU $50 ; High F (little bit lower than F)

TONE\_MID\_F EQU $60 ; Middle F

TONE\_LO\_F EQU $70 ; Low F

TONE\_VHI\_GSHARP EQU $80 ; Very High G# (G Sharp)

TONE\_HI\_GSHARP EQU $90 ; High G#

TONE\_MID\_GSHARP EQU $A0 ; Middle G#

TONE\_LO\_GSHARP EQU $B0 ; Low G#

TONE\_HI\_D EQU $C0 ; High D

TONE\_MID\_D EQU $D0 ; Middle D

TONE\_LO\_D EQU $E0 ; Low D

TONE\_PAUSE EQU $F0 ; Pause

;

; This is the default sound table

;

DEF\_SOUNDS

db SP\_1-SD\_1 ; 0000: 08

db SD\_1-DEF\_SOUNDS ; 0001: 0b BUTTON BEEP

db SD\_2-DEF\_SOUNDS ; 0002: 0c RETURN TO TIME

db SD\_3-DEF\_SOUNDS ; 0003: 0d HOURLY CHIME

db SD\_4-DEF\_SOUNDS ; 0004: 0e CONFIRMATION

db SD\_5-DEF\_SOUNDS ; 0005: 0f APPOINTMENT BEEP

db SD\_5-DEF\_SOUNDS ; 0006: 0f ALARM BEEP

db SD\_5-DEF\_SOUNDS ; 0007: 0f PROGRAM DOWNLOAD

db SD\_5-DEF\_SOUNDS ; 0008: 0f EXTRA

db SD\_6-DEF\_SOUNDS ; 0009: 11 COMM ERROR

db SD\_7-DEF\_SOUNDS ; 000a: 12 COMM DONE

;

; This is the [soundlet count table](#Soundlet_count_table) which contains the duration

; counts for the individual soundlets

;

SD\_1 db SND\_END+1 ; 000b: 81

SD\_2 db SND\_END+1 ; 000c: 81

SD\_3 db SND\_END+2 ; 000d: 82

SD\_4 db SND\_END+4 ; 000e: 84

SD\_5 db 10,SND\_END+24 ; 000f: 0a a8

SD\_6 db SND\_END+10 ; 0011: 8a

SD\_7 db SND\_END+16 ; 0012: a0

;

; This is the [soundlet pointer table](#Soundlet_pointer_table) which contains the pointers to the soundlets

;

SP\_1 db SL\_2-DEF\_SOUNDS ; 0013: 1d

SP\_2 db SL\_1-DEF\_SOUNDS ; 0014: 1b

SP\_3 db SL\_3-DEF\_SOUNDS ; 0015: 1f

SP\_4 db SL\_2-DEF\_SOUNDS ; 0016: 1d

SP\_5 db SL\_4-DEF\_SOUNDS ; 0017: 22

db SL\_5-DEF\_SOUNDS ; 0018: 27

SP\_6 db SL\_5-DEF\_SOUNDS ; 0019: 2a

SP\_7 db SL\_2-DEF\_SOUNDS ; 001a: 1d

;

; These are the [soundlets](#soundlet) themselves. The +1 or other number

indicates the duration for the sound.

;

SL\_1 db TONE\_HI\_GSHARP+1 ; 001b: 91

db TONE\_END ; 001c: 00

SL\_2 db TONE\_MID\_C+1 ; 001d: 31

db TONE\_END ; 001e: 00

SL\_3 db TONE\_MID\_C+2 ; 001f: 32

db TONE\_PAUSE+2 ; 0020: f2

db TONE\_END ; 0021: 00

SL\_4 db TONE\_HI\_C+2 ; 0022: 22

db TONE\_PAUSE+2 ; 0023: f2

db TONE\_HI\_C+2 ; 0024: 22

db TONE\_PAUSE+10 ; 0025: fa

db TONE\_END ; 0026: 00

SL\_5 db TONE\_HI\_C+2 ; 0027: 22

db TONE\_PAUSE+2 ; 0028: f2

db TONE\_END ; 0029: 00

SL\_6 db TONE\_HI\_C+3 ; 002a: 23

db TONE\_MID\_C+3 ; 002b: 33

db TONE\_END ; 002c: 00

;

; This is the tone that the comm app plays for each record

;

db TONE\_MIDC/16 ; 002d: 03

## Sound Files

The sound scheme stored in a file is nearly identical with the exception of a 4 byte header.  Given the default sound, you might picture it as below (with thanks to Pigeon for his first representation of this).



Brent Davidson gives a pretty good explanation of this:  (“Absolute offset” refer to the offset location in the file.  “Relative offset” refers to the location without the “header” (25 04 19 69).

The 08 at absolute offset 0004 indicates that the [soundlet count table](#Soundlet_count_table) is 8 bytes long.  In this case, we have only 7 different sounds, but one sound has two entries because it uses two [soundlets](#soundlet).

The next 10 bytes represent the relative offsets of the sound sequences. The relative offset of each byte reflects the system sound it represents. This table is fixed in size because there are only 10 [system sounds](#System_Sound).

The next 8 (or however many are indicated by absolute offset 0004) bytes (the soundlet count table) are in the relative offsets pointed to by the sound sequence table. The high order nibble of the byte indicates the last entry for this sound.  If it is clear, there are more soundlets associated with this sound.  The remaining 7 bits in the byte are the number of times that the corresponding soundlet is to be played.  Hence, a value of 0a indicates that the corresponding [soundlet](#soundlet) is to be played 10 times and the next entry in the [soundlet count table](#Soundlet_count_table) is to be used for the sound.  A value of 81 indicates that the corresponding [soundlet](#soundlet) is to be played once.

The next 8 bytes (or however many are indicated by absolute offset 0004) are the [soundlet pointer table](#Soundlet_pointer_table). They are parallel to the previous 8 bytes, and reference the relative offsets of the [soundlets](#soundlet).

The remainder of the bytes (except for the final byte) are the [soundlets](#soundlet) themselves. The high order nibble indicates the [tone](#tone), the low order nibble indicates the duration. A byte of 00 signals the end of each soundlet.

The low order nibble of the final byte of the file indicates the [tone](#tone) played after each record is downloaded during transmission, it’s high order nibble is always 0, and it’s count cannot be set.

# Wristapp Programming Reference

## The Processor

The Datalink contains a custom Motorola 6805 processor which performs all of the watch functions.  This turns out to be a very convenient thing as the 6805 is well documented and actually pretty fun to program (IMHO).  If you are looking for technical information, I tend to look to [Motorola’s 6805 home page](http://design-net.com/csic/CSIC_home.html) and to the instruction set card Oxford University Computing Laboratory’s [Microprocessor reference card](ftp://ftp.comlab.ox.ac.uk/pub/Cards/6805).  All of my work has been done with just these two information sources.

To summarize the 6805, it has two 8-bit registers (A and X) and a small number of addressing modes.  Since it has a 15 bit address bus, you are left with the interesting problem of using a register as a pointer.  To deal with this, you have to resort to self modifying code.  If you are only having to point to a small amount of memory, you can also use the indexed mode where the register is an offset from some base location.  Of course, if you only have to point to things in the first 256 bytes of ram, you can pretend that a register might be a pointer.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Bit Manip | | Branch | Read/Modify/Write | | | | | Control | | Register/Memory | | | | | |
|  | BTB | BSC | REL | DIR | INH | INH | INH | IX | INH | INH | IMM | DIR | EXT | IX2 | IX1 | IX |
|  | 0x | 1x | 2x | 3x | 4x | 5x | 6x | 7x | 8x | 9x | Ax | Bx | Cx | Dx | Ex | Fx |
| x0 | BRSET0 | BSET0 | BRA | NEG | NEGA | NEGX | NEGX | NEG | RTI |  | SUB | SUB | SUB | SUB | SUB | SUB |
| x1 | BRCLR0 | BCLR0 | BRN |  |  |  |  |  | RTS |  | CMP | CMP | CMP | CMP | CMP | CMP |
| x2 | BRSET1 | BSET1 | BHI |  |  |  |  |  |  |  | SBC | SBC | SBC | SBC | SBC | SBC |
| x3 | BRCLR1 | BCLR1 | BLS | COM | COMA | COMX | COMX | COM | SWI |  | CPX | CPX | CPX | CPX | CPX | CPX |
| x4 | BRSET2 | BSET2 | BCC | LSR | LSRA | LSRX | LSRX | LSR |  |  | AND | AND | AND | AND | AND | AND |
| x5 | BRCLR2 | BCLR2 | BCS |  |  |  |  |  |  |  | BIT | BIT | BIT | BIT | BIT | BIT |
| x6 | BRSET3 | BSET3 | BNE | ROR | RORA | RORX | RORX | ROR |  |  | LDA | LDA | LDA | LDA | LDA | LDA |
| x7 | BRCLR3 | BCLR3 | BEQ | ASR | ASRA | ASRX | ASRX | ASR |  | TAX |  | STA | STA | STA | STA | STA |
| x8 | BRSET4 | BSET4 | BHCC | LSL | LSLA | LSLX | LSLX | LSL |  | CLC | EOR | EOR | EOR | EOR | EOR | EOR |
| x9 | BRCLR4 | BCLR4 | BHCS | ROL | ROLA | ROLX | ROLX | ROL |  | SEC | ADC | ADC | ADC | ADC | ADC | ADC |
| xA | BRSET5 | BSET5 | BPL | DEC | DECA | DECX | DECX | DEC |  | CLI | ORA | ORA | ORA | ORA | ORA | ORA |
| xB | BRCLR5 | BCLR5 | BMI |  |  |  |  |  |  | SEI | ADD | ADD | ADD | ADD | ADD | ADD |
| xC | BRSET6 | BSET6 | BMC | INC | INCA | INCX | INCX | INC |  | RSP |  | JMP | JMP | JMP | JMP | JMP |
| xD | BRCLR6 | BCLR6 | BMS | TST | TSTA | TSTX | TSTX | TST |  | NOP | BSR\* | JSR | JSR | JSR | JSR | JSR |
| xE | BRSET7 | BSET7 | BIL |  |  |  |  |  | STOP |  | LDX | LDX | LDX | LDX | LDX | LDX |
| xF | BRCLR7 | BCLR7 | BIH | CLR | CLRA | CLRX | CLRX | CLR | WAIT | TXA |  | STX | STX | STX | STX | STX |

\* BSR Is a REL type instruction

INH - Inherent (1 Byte)

IMM - Immediate (2 Bytes) e.g. LDA #20

DIR - Direct (2 Bytes) e.g. LDA $61

EXT - Extended (3 Bytes) e.g. LDA $0244

REL - Relative (2 Bytes) e.g. BEQ \*+20

BSC - Bit Set/Clear (2 bytes)e.g. BSET2 $61

BTB - Bit test and Branch (3 bytes) e.g. BRCLR2 $61,\*+10

IX - Indexed (1 byte) e.g. ADD ,X or ADD 0,X

IX1 - Indexed 1 byte offset (2 bytes) e.g. LDA $61,X

IX2 - Indexed 2 byte offset (3 bytes) e.g. LDA $0122,X

## Tools

Unfortunately, there really aren’t a lot of tools out there for creating wristapps... While there are free assemblers available on [Motorola’s 6805 home page](http://design-net.com/csic/CSIC_home.html), you will find that the lack of support for Timex’s character set can be a bit limiting.  Even more problematic is that you have to figure out how to get the program to the watch in order to run it.

My solution has been to write my own assembler which creates the .zap file format that is understood by the Datalink software on the PC.  This [DLZap](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\dlzap.zip) program is pretty braindead in many ways and has quite a few bugs associated with refreshing the screen.  It also is limited to creating apps only for the 150 or the 150s one at a time.  If you want to create an app which runs on both watches, you have to combine them by hand.

I am working on a newer tool which doesn’t have the refresh bugs (yeah, right :-) and automatically creates both the 150 and the 150s applications.  Hopefully, this should be available in a couple of weeks.  (Like I ever got a chance to actually finish it , but read on :-).

## ASM6805 (2 months later)

Instead of fixing the refresh problems in DLZap, I realized that I needed something to address all of the work I was having to do to create wristApps and make it a bit easier (and hopefully more reproducible). I have gone and created a new version of the DLZap program which takes .zsm files and outputs the proper .zap file. Basically in a nutshell what it does is:

1. Compile from a single .zsm file and create both the 150 and the 150S versions of a Wristapp
2. Find the location where the Datalink software is installed and put the new wristapp there
3. Automatically update the timexdl.dat file to incorporate the wristapp
4. Integrate into Microsoft Developer studio to allow you to advance through errors with the F4 key.
5. Run as a windows app and allow you to select the file to assemble from a file requester

You can download the setup program for the beta [here](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\asm6805.zip).

## The .ZSM file Format

So, what is the .ZSM file format? It is nothing more than a standard .ASM file with a couple of comment lines at the beginning. For example the header for TIPCALC would be:

;Name: Tim Calculator

;Version: TIPCALC1

;Description: The tip calculator - by John A. Toebes, VIII

;

;Press the set button to enter the amount. When in set mode, press the MODE button to switch between dollars and cents mode.

;Press the set button to go back to the display mode. The tip amount will scroll across the bottom of the screen as 15%, 20% and then 10% in sequence.

;

;When in display mode, pressing the prev or next buttons will enter the set mode automatically on the dollars amount.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

The keywords are immediately after the semicolon and before the colon. The only recognized keywords are HEADER, NAME, VERSION, DESCRIPTION, HELPFILE, HELPTOPIC, and PARENT. It uses the VERSION keyword to identify the name of the created wristapp. The remaining information is just copied into the .ZAP file for use by the Timex software. In the process of doing this, I discovered that the last digit of the first line of a .ZAP file (the line that looks like TDL0405971) indicates whether the app is a 150-only app (last digit =1) or a 150/150S Dual app (last digit=2).

## System Routine Definitions

To compile a Wristapp, you need a header file which defines all of the routines that you might call.  For now, I have two versions of the same file - Wristapp.i which I put into two separate directories:

1. [Wristapp.i for the 150](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\Wristapp.zip)
2. [Wristapp.i for the 150s](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristapps.zip)

## Program Layout Basics

### Wristapp Interface Entries

Unlike more complex operating systems and modern programming environments, the Datalink Wristapps are simply a series of bytes to be loaded into the watch.  They are always loaded at $0110 and there is no relocation whatsoever.  This means that if you want to have more than one Wristapp in the watch at a time, you can’t.  However, you can get around this limitation by creating a Wristapp which performs more than one function.  The biggest issue with this will be the limited amount of ram ($0110 up $0436 minus however much you use for a sound scheme).  This works out to 804 bytes if you could have no sound scheme loaded.  Since the typical sound scheme is about 32 bytes, a more reasonable limit is 770 bytes for a wristapp - not a lot of room for sloppy code.

|  |  |  |
| --- | --- | --- |
| 0110 | WRIST\_MAIN | This is a JMP instruction to your primary initialization entry point for the wristapp. It is called immediately after the wristapp has been loaded for the first time and never again. |
| 0113 | WRIST\_SUSPEND | This is a JMP instruction to your suspend entry point.  It is called if your app is suspended because an alarm has gone off or your app has timed out because nothing has happened for 3 minutes.  If you don’t care about this, the three bytes should be a RTS followed by two NOP instructions. |
| 0116 | WRIST\_DOTIC | This is a JMP instruction to your callback handling routine.  It is called in any situation where the app has requested a callback for timed events such as the normal TIC (1/10th second), Second change, Minute change, Hour Change, and Day change.  If you do not want to handle these events, the three bytes should be a RTS followed by two NOP instructions |
| 0119 | WRIST\_INCOMM | This is a JMP instruction to your COMM suspend routine.  It is called when the COMM app wants to suspend your Wristapp which has requested a callback for timed events.  This gives your app a chance to forget about timers for a while.  Note that it is possible that the app may never be reentered if the user downloads a new wristapp on top of it.  If you don’t care about this, the three bytes should be a RTS followed by two NOP instructions. |
| 011C | WRIST\_NEWDATA | This is a JMP instruction to your new data handling routine.  It is called when the COMM app has downloaded new data to the watch.  This can be useful if you have an app that has to know about the data in the EEProm such as a password protect utility. If you don’t care about this, the three bytes should be a RTS followed by two NOP instructions. |
| 011F | WRIST\_GETSTATE | This is always two instructions:  LDA STATETAB,X  RTS  Which are used to get an entry from [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html).  The X register points to the entry that is to be retrieved.  You MUST supply this routine in order for the Wristapp to even function. |
| 0123 | WRIST\_JMP\_STATE0 | This is a JMP to the state 0 handling routine. |
| 0126 | WRIST\_OFF\_STATE0 | This is the offset into the state table for the state data associated with state 0.  Unless you reorder the states, this will always be 0. |
| 0127 | WRIST\_JMP\_STATE1 | This is a JMP to the state 1 handling routine (if any). |
| 012A | WRIST\_OFF\_STATE1 | This is the offset into the state table for the state data associated with state 1 (if any) |

This sequence of JMP instructions followed by the offset value repeats for all of the states that your Wristapp supports.  If you only have a single state, then your code can start at 0127.

### Strings and Data

With any typical program, you want to be able to write to the display.  If you can get away with using strings from the ROM, then you don’t have to worry about where to put the strings.  However, when you want to put your own strings there, you need to be aware that the BANNER8, PUT6TOP, and PUT6MID routines all take offsets from 0110 as the string to put on the display.  This effectively limits you to putting all of your strings at the start of the Wristapp. Since you also know that you can’t put a string until 0127, those first bytes of addressability are lost, limiting you to a total of 233 bytes of strings that you can store.

### .ZAP File Format

The Timex Datalink software on the PC stores all of the Wristapps in a .ZAP file.  The format of this file turns out to be pretty simple.  In fact, you can edit it using any standard text editor as long as you remember that the last line can not have a Carriage return after it.  This seems to make the Datalink software not always recognize the file.

Within the file, each section is terminated by a ¬ character ($AC).  You can optionally put a comment on the line immediately after the separator character.  For the V2.1 software, the .ZAP file contains the code for both the 150 and the 150s. For the earlier 2.0 software, the 150 code happens to be first and the 150s code is simply ignored.  This allows the same .zap file to work for both versions of the software.

|  |  |
| --- | --- |
| Applet file header | This is some sort of a version string associated with the creation time.  It is typically of the form “TDLmmddyyn” where mmddyy is the date that the applet was created and n is a sequence number.  The actual value of this string seems to be ignored. |
| Name 150 | This is the name of the applet as it is to appear in the Wristapps list for the 150.  The name can be any number of characters (there may be an upper limit on it) and can contain spaces and other special characters. |
| Version 150 | This is the version number of the 150 applet.  It should be up to 8 characters of alphanumeric characters.  It is not clear that this is actually used by the software. |
| Description150 | This is the description for the 150 applet that is shown when you select it in the Wristapp panel.  The description can be pretty much any length and even include blank lines.  The software does its best to wrap this description when it displays it. |
| Help Filename 150 | This is the name of the Windows .hlp file that is to be used when the user asks for help on the 150 applet. The default file that timex uses for all of its wristapps is WATCHAPP.HLP.  You should provide a .hlp file for any wristapp which tells the user how the Wristapp works on the watch. |
| Help Index 150 | This is the index in the help file associated with the help for the 150 applet.  This is passed along with the Help Filename to the Windows Help system. |
| Config App 150 | This is the configuration program (if any) that is to be invoked when the user selects the configure button  in the Wristapps software.  This program should be a standalone Windows program that modifies the applet as appropriate.  If the program is not configurable, the string should be “none” |
| Watch 150 | This is the name of the watch that this applet is targeted at.  It should be “Timex Data Link 150 Watch” |
| Code 150 | This is the hex code for the 150 applet.  It is simply the ASCII dump of the hex digits (0-9A-Z) of the code to be downloaded to the watch.  It really should be a single line of text with no spaces, but it does appear to allow the line to wrap.  Since the longest this line can ever be is 1608 characters, there really isn’t any need to wrap the line. |
| CRC 150 | This is the CRC-16 associated with the 150 applet. It is only a CRC on the Code 150 string. |
| Data Indicator 150 | This is the indicator of data for the 150 applet.  If there is no data, this should be a 0, otherwise it is a 1. |
| Data 150 | (OPTIONAL) This is the data for the 150 applet. This entry is present ONLY if the Data Indicator 150 value is 1. |
| Name 150s | This is the name of the applet as it is to appear in the Wristapps list for the 150s.  The name can be any number of characters (there may be an upper limit on it) and can contain spaces and other special characters. |
| Version 150s | This is the version number of the 150s applet.  It should be up to 8 characters of alphanumeric characters.  It is not clear that this is actually used by the software. |
| Description150s | This is the description for the 150s applet that is shown when you select it in the Wristapp panel.  The description can be pretty much any length and even include blank lines.  The software does its best to wrap this description when it displays it. |
| Help Filename 150s | This is the name of the Windows .hlp file that is to be used when the user asks for help on the 150s applet. The default file that Timex uses for all of its wristapps is WATCHAPP.HLP.  You should provide a .hlp file for any wristapp which tells the user how the Wristapp works on the watch. |
| Help Index 150s | This is the index in the help file associated with the help for the 150 applet.  This is passed along with the Help Filename to the Windows Help system. |
| Config App 150s | This is the configuration program (if any) that is to be invoked when the user selects the configure button  in the Wristapps software.  This program should be a standalone Windows program which modifies the applet as appropriate.  If the program is not configurable, the string should be “none” |
| Watch 150s | This is the name of the watch that this applet is targeted at.  It should be “Timex Data Link 150s Watch” |
| Code 150s | This is the hex code for the 150s applet.  It is simply the ASCII dump of the hex digits (0-9A-Z) of the code to be downloaded to the watch.  It really should be a single line of text with no spaces, but it does appear to allow the line to wrap.  Since the longest this line can ever be is 1608 characters, there really isn’t any need to wrap the line. |
| CRC 150s | This is the CRC-16 associated with the 150s applet. It is only a crc on the Code 150s string. |
| Data Indicator 150s | This is the indicator of data for the 150s applet.  If there is no data, this should be a 0, otherwise it is a 1. |
| Data 150s | (OPTIONAL) This is the data for the 150s applet. This entry is present ONLY if the Data Indicator 150 value is 1. |

## Getting Started

When your program is first invoked, you have to set a bit to tell the Roms that you are ready to handle processing.  To do this, you need to set bit 7 in the WRISTAPP\_FLAGS ($96).  At this time, you probably want to set a few of the other requests to indicate how your Wristapp wants to process things.  The bits in this flag byte are interpreted as:

###### WRISTAPP\_FLAGS - $96

|  |  |  |
| --- | --- | --- |
| 7 | Wristapp has been loaded | SET=LOADED |
| 6 | Uses system rules for button beep decisions | SET=System Rules |
| 5 | Play button beep sound on wristapp for mode button | SET=ENABLE |
| 4 | Play button beep sound on wristapp for any button | SET=ENABLE |
| 3 | wristapp wants a call once a day when it changes (WRIST\_DOTIC) | SET=CALL |
| 2 | wristapp wants a call once an hour when it changes (WRIST\_DOTIC) | SET=CALL |
| 1 | wristapp wants a call once a minute when it changes (WRIST\_DOTIC) | SET=CALL |
| 0 | wristapp wants a second timer function called at start of interrupt (WRIST\_DOTIC) | SET=CALL |

## The State Table

An app is generally run through events passed in to it. These events are controlled by a series of state tables which indicate which events are to put the app into what state and how long to process that app for. A state table consists of a single byte followed by a series of three byte entries with a [EVENT\_END](#EVENT_END) terminator byte after the last entry. Each entry has three parts to it:

1. The event code which indicates what event is to be accepted by this state table
2. The timer indicator to indicate how long to wait before firing off a timer if no other event occurs before it.  The values can be found in the [table](#Timer_Constants) below
3. The new state to enter when this event is encountered

The initial byte is the state to enter if an event is encountered which does not match any entry in the table.

### Special State Tables

State table 0 is always entered first for an app. It will almost always have an [EVT\_ENTER](#EVT_ENTER) entry in it so that you can know when an application is first called.

If an app supports nesting (all WristApps might), then it will be entered by a call to State Table 1 with an [EVT\_NEST](#EVT_NEST) event. All other state tables are completely defined by the application and may be used in any way that you want. Often a separate state is used for each mode that the app might have (such as a set mode). In order to switch between states, either you code the new state with the event, such as with the [EVT\_SET](#EVT_SET) operation OR you can post a user event which has an associated entry in the state table that has the new state for that event.

There are two special state values associated with an event. $FF is used to indicate that the app wishes to exit and go to the next app. For WristApps, this means go back to the time app. $FE is a special value used to handle returning from a [EVT\_NEST](#EVT_NEST) nesting. If all of the nested app processing occurs in state 1, then this value would appear for an entry in the state1 table. For all others, it is assumed to be the new state table to select. No error checking is done on any of these values.

One very nice thing that can be done with the events is posting a timer to go off if no other event occurs after the current event. There are two timers although only one can be active at a time. The reason for this is to allow the app to quickly distinguish between which event timed out without having to save some global variable. These timer values are fixed in the ROM and you select which timer interval you want through the value you set. For a strange happenstance, all of the intervals of the second timer are also available for the first time (but I would be careful not to count on that).

### Nested Apps

One important event that an application should handle is the [EVT\_RESUME](#EVT_RESUME) which occurs after a nested app terminates. This allows your application to pick up after an alarm or appointment has gone off. When you get this event, it is a pretty good idea to refresh the display since you don’t know what state the other app left it in. You should also use this time to restore any system flags that you may have set. You should also be aware that before your app is suspended, the system will call your suspend function at [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND) ($0113). That will be your chance to save any variables that you expect to have trashed.

### Button Events

For the events, there are three forms of the button events. The [EVT\_NEXT](#EVT_NEXT), [EVT\_MODE](#EVT_MODE), [EVT\_SET](#EVT_SET), [EVT\_PREV](#EVT_PREV), and [EVT\_GLOW](#EVT_GLOW) events allow you to see when the corresponding button is pressed. When you get one of these events, you will not get notification of when the button was released. There is a set of events [EVT\_DNNEXT](#EVT_DNNEXT), [EVT\_DNMODE](#EVT_DNMODE), [EVT\_DNSET](#EVT_DNSET), [EVT\_DNPREV](#EVT_DNPREV) and [EVT\_DNGLOW](#EVT_DNGLOW) which give you the down transition for those buttons and the corresponding set of events [EVT\_UPNEXT](#EVT_UPNEXT), [EVT\_UPMODE](#EVT_UPMODE), [EVT\_UPSET](#EVT_UPSET) (I like that name), [EVT\_UPPREV](#EVT_UPPREV), and [EVT\_UPGLOW](#EVT_UPGLOW) which tell you when the button has been released. It is the case that the UP event can be handled by a different state than the DN event.

If you want to get any of those buttons, you can look for [EVT\_ANY](#EVT_ANY) (and [EVT\_DNANY](#EVT_DNANY), [EVT\_UPANY](#EVT_UPANY)) which will call when any of the 5 buttons have been pressed. In order to figure out which button was pressed, your code will need to look at BTN\_PRESSED ($04c3) which will contain one of the [EVT\_NEXT](#EVT_NEXT), [EVT\_MODE](#EVT_MODE), [EVT\_SET](#EVT_SET), [EVT\_PREV](#EVT_PREV), and [EVT\_GLOW](#EVT_GLOW) values. Often an application does not have an interest in the Indiglo button but cares about the other 4 buttons. For this, you can use [EVT\_ANY4](#EVT_ANY4) (and [EVT\_DNANY4](#EVT_DNANY4), [EVT\_UPANY4](#EVT_UPANY4)) just the same way as the [EVT\_ANY](#EVT_ANY) events.

### Timer Events

The [EVT\_TIMER1](#EVT_TIMER1) and [EVT\_TIMER2](#EVT_TIMER2) events come in when the timer associated with a particular event has elapsed without another event being posted. There is no requirement of using a particular timer for a given event other than to allow you to distinguish between which event occurred. The two timers have slightly different values for when they go off and that might slightly affect your choice of timers (but that is rare).  From experimentation, it appears that the time cycle for the TIMER1 is a bit slower than that for Timer2.  I recommend that you use Timer2 for any of the fast actions and timer1 for the slower ones (like timing out the display).

### Other Events

The [EVT\_USER0](#EVT_USER0), [EVT\_USER1](#EVT_USER1), [EVT\_USER2](#EVT_USER2), and [EVT\_USER3](#EVT_USER3) events are for an application to use for anything it wants to. Most of the time, these are useful for transitioning to a different state. You can post an event by calling [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT).

The only other event is [EVT\_IDLE](#EVT_IDLE). This event is sent only to the TIME app when another app has been suspended because it was idle for more than three minutes.  Since a wristapp could never get this event, it is probably worth ignoring.

### Event Constants

Here are the constants which you would find useful in creating your app:

#### State Table Values

|  |  |  |
| --- | --- | --- |
| EVT\_NEXT | $00 | Next button pressed (not interested in the up transition) |
| EVT\_MODE | $01 | Mode button pressed (not interested in the up transition) |
| EVT\_SET | $02 | Set/Delete button pressed (not interested in the up transition) |
| EVT\_PREV | $03 | Prev button pressed (not interested in the up transition) |
| EVT\_GLOW | $04 | Indiglo button pressed (not interested in the up transition) |
| EVT\_ANY | $05 | Any button pressed (not interested in the up transition) |
| EVT\_ANY4 | $06 | Any button pressed except Indiglo (not interested in the up transition) |
| EVT\_IDLE | $19 | This is only sent to the TIME app when another app has been idle for more than three minutes |
| EVT\_RESUME | $1a | Called when resuming from a nested app |
| EVT\_ENTER | $1b | Initial state. |
| EVT\_NEST | $1c | The state table 1 entry called when a nested application is called. It is the equivalent of EVT\_ENTER for an interrupt. This only occurs for WristApps, Timer, and appt apps. |
| EVT\_END | $1d | End of event table indicator |
| EVT\_TIMER1 | $1e | Timer event - This is fired for the TIM1\_ values |
| EVT\_TIMER2 | $1f | Timer event - This is fired for the TIM2\_ values |
|  | $20-$36 | UNUSED (I bet that you can have user specified events for these too) |
| EVT\_USER0 | $37 | User specified events. Queued by calling [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT) |
| EVT\_USER1 | $38 | User specified events. Queued by calling [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT) |
| EVT\_USER2 | $39 | User specified events. Queued by calling [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT) |
| EVT\_USER3 | $3a | User specified events. Queued by calling [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT) |
|  | $3b-$7f | UNUSED |
| EVT\_DNNEXT | $80 | Next button pressed |
| EVT\_DNMODE | $81 | Mode button pressed |
| EVT\_DNSET | $82 | Set/Delete button pressed |
| EVT\_DNPREV | $83 | Prev button pressed |
| EVT\_DNGLOW | $84 | Indiglo button pressed |
| EVT\_DNANY | $85 | Any of the four buttons Pressed |
| EVT\_DNANY4 | $86 | Any button pressed except Indiglo |
|  | $87-$9F | UNUSED |
| EVT\_UPNEXT | $A0 | Next button released |
| EVT\_UPMODE | $A1 | Mode button released |
| EVT\_UPSET | $A2 | Set/Delete button released |
| EVT\_UPPREV | $A3 | Prev button released |
| EVT\_UPGLOW | $A4 | Indiglo button released |
| EVT\_UPANY | $A5 | Any of the four buttons Released |
| EVT\_UPANY4 | $A6 | Any button Released except Indiglo |

#### Timer Constants

|  |  |  |
| --- | --- | --- |
| TIM\_ONCE | $ff | No time interval. Operation is executed just once |
| TIM1\_TIC | $00 |  |
| TIM1\_2TIC | $01 |  |
| TIM1\_3TIC | $02 |  |
| TIM1\_4TIC | $03 |  |
| TIM1\_HALFSEC | $04 |  |
| TIM1\_SECOND | $05 |  |
| TIM1\_SECHALF | $06 |  |
| TIM1\_TWOSEC | $07 |  |
| TIM1\_TWOSEC1 | $08 |  |
| TIM1\_12SEC | $09 |  |
| TIM1\_18SEC | $0a |  |
| TIM2\_TIC | $80 | This is the typical scroll interval |
| TIM2\_2TIC | $81 |  |
| TIM2\_4TIC | $82 |  |
| TIM2\_8TIC | $83 | This is the normal blink interval |
| TIM2\_12TIC | $84 | Just over a second |
| TIM2\_16TIC | $85 | A second and a half |
| TIM2\_24TIC | $86 | Two and a half seconds |
| TIM2\_32TIC | $87 | Just over three seconds |
| TIM2\_40TIC | $88 | Four seconds |
| TIM2\_48TIC | $89 | Almost five seconds |
| TIM2\_96TIC | $8a | Almost ten seconds |

Note that the second part of this table is happen-stance since it is really a rollover of the second table on top of the first one. But it might be useful to someone...

|  |  |  |
| --- | --- | --- |
| TIM1\_TICA | $0b | This is the typical scroll interval |
| TIM1\_2TICA | $0c |  |
| TIM1\_4TICA | $0d |  |
| TIM1\_8TIC | $0e | This is the normal blink interval |
| TIM1\_12TIC | $0f | Just over a second |
| TIM1\_16TIC | $10 | A second and a half |
| TIM1\_24TIC | $11 | Two and a half seconds |
| TIM1\_32TIC | $12 | Just over three seconds |
| TIM1\_40TIC | $13 | Four seconds |
| TIM1\_48TIC | $14 | Almost five seconds |
| TIM1\_96TIC | $15 | Almost ten seconds |

## Classes of Callable Functions

I have broken down the system routines into 14 basic categories.  For each function listed, you will find the name of the routine followed by two hex addresses separated by a slash.  The first address is the location of the routine for the Datalink 150 and the second is the location for that routine on the 150s.

|  |  |
| --- | --- |
| [Anniversary support](#Anniversary) | General routines for accessing the Anniversary data in the EEProms and setting all of the flags and display to indicate the anniversaries. |
| [Appointment support](#Appointment) | General routines for accessing the appointment data in the EEProms and setting all of the flags and display segments for appointments. |
| [Blinking routines](#Blinking) | ???? |
| [Event support](#Event) | ???? |
| [Format Routines](#Format) | Routines for converting numbers into the corresponding display digits. |
| [Indiglo support](#Indiglo) | Routines for turning on and off the Indiglo light as well as managing the timers for the light |
| [INST Support](#INST) | ???? |
| [Internal](#Internal) | Not quite sure why you would ever call these routines, but the MIGHT be useful sometimes. |
| [Line routines](#Line) | ???? |
| [Packet/EEProm Support](#Packet) | ???? |
| [Scanning support](#Scanning) | ???? |
| [Scrolling Messages](#Scrolling) | ???? |
| [Sound Support](#Sound) | ???? |
| [Update functions](#Update) | ???? |

### Anniversary support routines

|  |  |
| --- | --- |
| Routine | **FIND\_ANNIV\_TODAY** - $40CD/$40BC |
| Parameters | None |
| Purpose | This finds the next anniversary entry which is greater than or equal to today |

|  |  |
| --- | --- |
| Routine | **FIND\_ANNIV\_SCAN** - $40D3/$40C2 |
| Parameters | ANNIVSCAN\_MONTH, ANNIVSCAN\_YEAR, ANNIVSCAN\_DAY - Date to scan for anniversary entry |
| Purpose | This finds the next anniversary entry which is greater than or equal to the scan date |

|  |  |
| --- | --- |
| Routine | **ANNIV\_NEXT\_ENTRY** - $40E1/$40D0 |
| Parameters | ANNIV\_CURRENT – The current anniversary entry |
| Purpose | Advance to the next anniversary entry. If we hit the end of the list, we need to wrap the year and go to the next one |

|  |  |
| --- | --- |
| Routine | **ANNIV\_PREV\_ENTRY** - $4117/$4106 |
| Parameters | ANNIV\_CURRENT – The current anniversary entry |
| Purpose | Advance to the previous anniversary entry. If we hit the end of the list, we need to wrap the year and go to the end again |

|  |  |
| --- | --- |
| Routine | **FIND\_ANNIV\_ENTRY** - $415F/$414E |
| Parameters | ANNIVTEST\_MONTH, ANNIVTEST\_DAY, ANNIVTEST\_YEAR - Date of anniversary to find |
| Purpose | This finds the next anniversary entry which is greater than or equal to the specified date |

|  |  |
| --- | --- |
| Routine | **CHECK\_ANNIVERSARIES** - $41FC/$41EB |
| Parameters | None |
| Purpose | This code checks all anniversaries to see if any occur today |

|  |  |
| --- | --- |
| Routine | **SET\_ANNIVTEST\_TODAY** - $423A/$4229 |
| Parameters | None |
| Purpose | Latches the current month, date, year into the ANNIVTEST\_ locations |

|  |  |
| --- | --- |
| Routine | **INIT\_ANNIVERSARY\_DATA** - $4282/$4271 |
| Parameters | None |
| Purpose | This clears the ANNIVERSARY occurrence flags and latches in the current date for the anniversary check routine |

|  |  |
| --- | --- |
| Routine | **TEST\_ANNIVERSARY** - $4288/$4277 |
| Parameters | EXTRACTBUF – Anniversary data to be checked ANNIVTEST\_MONTH, ANNIVTEST\_DAY, ANNIVTEST\_YEAR - Current date to check against |
| Purpose | This tests the anniversary against the current day and sets the 4,ANNIV\_FLAGS and 5,ANNIV\_FLAGS flags appropriately. |

|  |  |
| --- | --- |
| Routine | **ANNIV\_COPY\_INFO** - $4308/$42F7 |
| Parameters | ANNIV\_YEAR - The year to fake the appointment as |
| Purpose | This copies the current appointment information into the ANNIVSCAN variables |

|  |  |
| --- | --- |
| Routine | **READ\_ANNIV\_CURRENT** - $4317/$4306 |
| Parameters | ANNIV\_CURRENT – the anniversary entry to be read |
| Purpose | This reads in the current anniversary entry into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **READ\_ANNIV\_FIRST** - $4326/$4315 |
| Parameters | None |
| Purpose | This reads the first anniversary entry into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **READ\_ANNIV\_NEXT** - $4335/$4324 |
| Parameters | None |
| Purpose | This reads the next anniversary entry into EXTRACTBUF |

### Scanning support

|  |  |
| --- | --- |
| Routine | **TEST\_SCAN\_START** - $4346/$4335 |
| Parameters | SCAN\_MONTH - Month, Day, Year of appointment to compare SCAN\_DAY SCAN\_YEAR TMAPP\_MONTH - Current Month, Day, Year TMAPP\_DAY TMAPP\_YEAR |
| Purpose | Sets 0,SCAN\_FLAGS to indicate that the current scan date is out of range. |

|  |  |
| --- | --- |
| Routine | **FIX\_SCAN\_YEAR** - $4371/$4360 |
| Parameters | SCAN\_YEAR - Year to be adjusted |
| Purpose | Adjusts SCAN\_YEAR to account for years past 2000 |

|  |  |
| --- | --- |
| Routine | **TEST\_SCAN\_END** - $437E/$436D |
| Parameters | SCAN\_MONTH, SCAN\_DAY, SCAN\_YEAR - Current scan date SCAN\_END\_MONTH, SCAN\_END\_DAY, SCAN\_END\_YEAR - Limit of the scan range |
| Purpose | Tests to see if the current scan date is past the end range for the scan. If so, it sets 0,SCAN\_FLAGS |

|  |  |
| --- | --- |
| Routine | **RESTORE\_SCAN\_YEAR** - $43AE/$439D |
| Parameters | SCAN\_YEAR - Year to be adjusted |
| Purpose | Restores SCAN\_YEAR to be in the 0-99 range (After a call to FIX\_SCAN\_YEAR) |

|  |  |
| --- | --- |
| Routine | **INCREMENT\_SCAN\_DATE** - $43B9/$43A8 |
| Parameters | SCAN\_MONTH, SCAN\_DAY, SCAN\_YEAR |
| Purpose | Increments the current scan day by one |

|  |  |
| --- | --- |
| Routine | **GET\_SCAN\_MONTHLEN** - $43E0/$43CF |
| Parameters | None |
| Purpose | This computes the end of the month based on SCAN\_MONTH and SCAN\_YEAR |

|  |  |
| --- | --- |
| Routine | **DECREMENT\_SCAN\_DATE** - $43F4/$43E3 |
| Parameters | SCAN\_MONTH, SCAN\_YEAR |
| Purpose | Decrements the scan data by one |

### Appointment support

|  |  |
| --- | --- |
| Routine | **FIND\_APPT\_NOW** - $4415/$4404 |
| Parameters | None |
| Purpose | This finds and reads in an appointment which will occur next after the current time in the current time zone. The appointment is put into EXTRACTBUF and all appropriate variables are set. |

|  |  |
| --- | --- |
| Routine | **FIND\_APPT\_SCAN** - $441B/$440A |
| Parameters | SCAN\_MONTH,DAY,YEAR |
| Purpose | This finds and reads in an appointment which will occur next after the current scan values. The appointment is put into EXTRACTBUF and all appropriate variables are set. |

|  |  |
| --- | --- |
| Routine | **SET\_APPTFIND\_SCAN** - $4422/$4411 |
| Parameters | SCAN\_MONTH, SCAN\_DAY, SCAN\_YEAR |
| Purpose | This copies over the current SCAN variables into the APPTFIND variables |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_NEXT** - $442C/$441B |
| Parameters | APPT\_CURRENT, APPT\_LAST - current and last appointment entries |
| Purpose | This reads in the next appointment into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **APPT\_LATCH\_ENTRYDATA** - $4468/$4457 **APPT\_LATCH\_ENTRYONLY** - $446C/$445B |
| Parameters | EXTRACTBUF - current appointment entry APPTEST\_YEAR - year of the entry |
| Purpose | These copy the current appointment data into the corresponding system variables The ENTRYONLY routine doesn’t copy over the year because it presumably has already been copied. |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_PREV** - $447C/$446B |
| Parameters | APPT\_CURRENT, APPT\_LAST - current and last appointment entries |
| Purpose | This reads in the previous appointment into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **FIND\_APPT\_ENTRY** - $44C6/$44B5 |
| Parameters | APPTFIND\_YEAR,DAY,MONTH,QHOUR,HOUR |
| Purpose | This finds an appointment that matches or exceeds the APPTFIND values |

|  |  |
| --- | --- |
| Routine | **APPT\_LATCH\_ENTDYDATA** - $45A5/$4594 |
| Parameters | APPT\_ENTRY - Entry to latch appointment information for |
| Purpose | This copies the current appointment entry into the corresponding system variables so that we can continue comparing appointments |

|  |  |
| --- | --- |
| Routine | **CHECK\_APPOINTMENTS** - $45B9/$45A8 |
| Parameters | APPT\_QHOUR\_NOW - The current quarter-hour APPT\_BASEYEAR - The base year for the first appointment |
| Purpose | This tests to see if any appointments are ready to go off. It posts a nested app for any appointments |

|  |  |
| --- | --- |
| Routine | **SET\_APPTFIND\_NOW** - $462A/$4619 |
| Parameters | None |
| Purpose | Sets the appointment find variables to the current time |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_FIRST** - $4686/$4675 |
| Parameters | APPT\_FIRST |
| Purpose | Read in the first appointment |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_LAST** - $469D/$468C |
| Parameters | APPT\_LAST - the entry of the last appointment |
| Purpose | This reads in the last appointment entry |

|  |  |
| --- | --- |
| Routine | **CHECK\_APPT\_TIME** - $46B7/$46A6 |
| Parameters | None |
| Purpose | This checks to see if any appointments are ready to go off |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_PACKET1** - $473A/$4729 |
| Parameters | None |
| Purpose | This reads the first appointment packet into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **READ\_NEXT\_APPT\_PACKET** - $4749/$4738 |
| Parameters | None |
| Purpose | This reads in the next appointment packet into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **READ\_APPT\_CURRENT** - $475A/$4749 |
| Parameters | APPT\_CURRENT - the appointment entry to be read |
| Purpose | This reads in the current appointment entry into EXTRACTBUF |

### Internal

|  |  |
| --- | --- |
| Routine | **ANNIV\_GETMONTHLEN** - $426A/$4259 |
| Parameters | ANNIV\_MONTH – Month to calculate ANNIVTEST\_YEAR – Year to calculate |
| Purpose | This computes the number of days in the given month |

|  |  |
| --- | --- |
| Routine | **ACQUIRE\_TIME** - $4F22/$4F11 |
| Parameters | None |
| Purpose | This acquires the right to change the time. All alarms and anniversaries will temporarily be ignored until RELEASE\_TIME has been called |

|  |  |
| --- | --- |
| Routine | **RELEASE\_TIME** - $4F2E/$4F1D |
| Parameters | None |
| Purpose | This releases the lock on time and allows all alarms and anniversaries to be checked once again. |

### Indiglo support

|  |  |
| --- | --- |
| Routine | **QUEUE\_INDIGLO\_OFF** - $49D9/$4C38 |
| Parameters | None |
| Purpose | Queue up the timer for shutting off the Indiglo if the Indiglo is enabled and we are in night mode. |

|  |  |
| --- | --- |
| Routine | **INDIGLO\_OFF** - $4E8E/$4E7D |
| Parameters | None |
| Purpose | This routine turns off the Indiglo light |

|  |  |
| --- | --- |
| Routine | **NIGHTMODE\_INDIGLO\_ON** - $49E6/$4C45 |
| Parameters | None |
| Purpose | Queue up the timer for shutting off the Indiglo if the Indiglo is enabled and we are in night mode. The INDIGLO\_ON routine just simply turns the Indiglo on immediately |

|  |  |
| --- | --- |
| Routine | **INDIGLO\_ON** - $49EC/$4C4B |
| Parameters | None |
| Purpose | Queue up the timer for shutting off the Indiglo if the Indiglo is enabled and we are in night mode. The INDIGLO\_ON routine just simply turns the Indiglo on immediately |

### Sound Support

|  |  |
| --- | --- |
| Routine | **SNDSTART** - $4E4A/$4E39 |
| Parameters | SYSSOUND - Current sound to be playing |
| Purpose | Start playing the current sound in SYSSOUND |

|  |  |
| --- | --- |
| Routine | **STOP\_ALL\_SOUND** - $4E68/$4E57 |
| Parameters | None |
| Purpose | Keep the sound hardware running or reset everything else |

|  |  |
| --- | --- |
| Routine | **PLAYCONF** - $4E7A/$4E69 |
| Parameters | None |
| Purpose | Play a confirmation sound |

|  |  |
| --- | --- |
| Routine | **PLAYBUTTON** - $4E80/$4E6F |
| Parameters | None |
| Purpose | Play the button beep sound if no other sound is currently playing |

|  |  |
| --- | --- |
| Routine | **PLAY\_HOURLY** - $4EB1/$4EA0 |
| Parameters | None |
| Purpose | Plays the hourly sound if nothing else is playing and sounds are enabled |

|  |  |
| --- | --- |
| Routine | **SNDSTOP** - $4F3A/$4F29 |
| Parameters | None |
| Purpose | This stops whatever sound is currently playing |

|  |  |
| --- | --- |
| Routine | **PLAY\_BUTTON\_SAFE** - $4F46/$4F35 |
| Parameters | None |
| Purpose | This will play the button beep sound if it hasn’t just been played |

### Event support

|  |  |
| --- | --- |
| Routine | **POSTEVENT** - $4E89/$4E78 |
| Parameters | A - Event to be posted. |
| Purpose | Post a event to the internal processing queue This posts an event to run through the processing loop for the current applet. Typical user events are in the $30-$3F range. |

|  |  |
| --- | --- |
| Routine | **CALL\_NESTEDAPP** - $4F4D/$4F3C |
| Parameters | A - Nested application number. This is one of the three defined apps: 9 = APP2\_ALARM - Alarm (while another app is running) 10 = APP2\_APPT - Appointment (while another app is running) 11 = APP2\_WRIST - Wristapp (while another app is running) X - Parameter to pass to the nested application |
| Purpose | This sets up to call a nested application while the current one is running. Up to 5 apps may be nested (although there are only 3 potential ones defined). If more than 5 have been called the oldest one will be forgotten. When the nested app is called, NESTED\_APP will be set to the application number passed in and NESTED\_PARM will contain the X parameter passed in |

### Packet/EEProm Support

|  |  |
| --- | --- |
| Routine | **UNPACK\_PHONENUM** - $4FBF/$4FAE |
| Parameters | EXTRACTBUF+1 – Pointer to 6 bytes of compressed phone number information |
| Returns | BUF\_PHONENUM – Contains 12 byte unpacked number |
| Purpose | This gets a compressed phone number and puts it in the phone number buffer Phone numbers are compressed into nibbles instead of bytes, allowing a number to be packed in half the space. As a result, a number can contain only 16 possible characters: “01234567890CFHPW “ Any other characters are encoded as a space before being sent down. The presumption is that the characters allow for the number and indicators for: Cell Fax Home Pager Work |

|  |  |
| --- | --- |
| Routine | **UNPACK\_STRING** - $4FF0/$4FDF |
| Parameters | PARM\_UNPACKOFF - Offset into the start of the compressed buffer EXTRACTBUF - packed data |
| Returns | MSGBUF - Contains the unpacked string |
| Purpose | This gets a compressed string and puts it into the scrolling message buffer Strings are packed 6 bits across so that 4 unpacked characters can fit in 3 bytes This routine will unpack enough bits so that the resultant message buffer is exactly 32 bytes long. It is assumed that the end of the buffer message is stored in the packed string. |

|  |  |
| --- | --- |
| Routine | **READ\_PACKET** - $503E/$502D |
| Parameters | PARM\_LEN - Number of bytes to copy PARM\_PACKET – Packet number to read X - Packet group to search (0,2,4,6) 0 = APPT Entries 2 = List entries 4 = Phone Number 6 = Anniversaries |
| Purpose | Reads the requested packet into EXTRACTBUF |

|  |  |
| --- | --- |
| Routine | **FIND\_PACKET** - $5044/$5033 |
| Returns | INST\_ADDRHI:INST\_ADDRLO - points to the start of the packet |
| Parameters | PARM\_PACKET - Packet number to locate X - Packet group to search (0,2,4,6) 0 = APPT Entries 2 = List entries 4 = Phone Number 6 = Anniversaries |
| Purpose | This advances to the given packet in the packet group |

|  |  |
| --- | --- |
| Routine | **DO\_TRANSFER** - $505F/$504E |
| Parameters | PARM\_LEN - Number of bytes to copy INST\_ADDRHI - Address of source data to copy |
| Purpose | This transfers the data from the indicated location to EXTRACTBUF The source can be the EEPROM or somewhere else in memory |

|  |  |
| --- | --- |
| Routine | **TOGGLE\_ENTRYFLAG** - $5077/$5066 |
| Parameters | None |
| Purpose | This toggles the high bit of the first byte in an entry |

|  |  |
| --- | --- |
| Routine | **INIT\_EEPROMPOINTERS** - $5080/$506F |
| Parameters | None |
| Purpose | Initializes all of the EEProm data pointers to reflect empty data for all of the applications |

|  |  |
| --- | --- |
| Routine | **RESET\_EEPROMENTRIES** - $508D/$507C |
| Parameters | None |
| Purpose | Re-Initializes all of the EEProm data pointers to reflect empty data for all of the applications |

|  |  |
| --- | --- |
| Routine | **REINIT\_APP\_DATA** - $50A7/$5096 |
| Parameters | None |
| Purpose | This routine is called after new data has been loaded into the EEPROM |

### INST Support

|  |  |
| --- | --- |
| Routine | **MAKE\_INST\_LDA** - $50B4/$50A3 **MAKE\_INST\_LDA\_X** - $50B8/$50A7 **MAKE\_INST\_STA** - $50BC/$50AB |
| Parameters | None |
| Purpose | These routines make the INST2 opcodes to be an LDA or STA $nnnn,X instruction |

|  |  |
| --- | --- |
| Routine | **ADD\_INSTADDR** - $50C7/$50B6 |
| Parameters | A - value to add to the current INST\_ADDR base address |
| Purpose | This takes an offset value and subsumes it into the already constructed instruction starting at INST\_OPCODE |

|  |  |
| --- | --- |
| Routine | **SET\_INSTADDR\_0110** - $50D7/$50C6 |
| Parameters | X – R |
| Purpose | This routine sets INST\_ADDRHI:INST\_ADDRLO to be 0110 |

|  |  |
| --- | --- |
| Routine | **GET\_INST\_BYTE** - $50EB/$50DA |
| Parameters | INST\_ADDRHI:INST\_ADDRLO - the pointer to the byte to get |
| Purpose | This routine gets the single byte from the indicated location either in the EEPROM or somewhere in memory. |

|  |  |
| --- | --- |
| Routine | **WRITE\_FLAG\_BYTE** - $510A/$50F9 |
| Parameters | INST\_ADDRHI:INST\_ADDRLO - the pointer to the byte to write to |
| Purpose | This routine writes a single byte to the indicated location either in the EEPROM or somewhere in memory. ?????? This adjusts an address relative to the Sound buffers. |
| Parameters | A - Offset into the sound data area |

|  |  |
| --- | --- |
| Routine | **FILL\_EXTRACTBUF** - $513E/$512D |
| Parameters | PARM\_LEN - Number of bytes to be copied INST\_ADDRHI:INST\_ADDRLO - Address in Prom to read |
| Returns | EXTRACTBUF – Contains the bytes read in from the EEPROM |
| Purpose | This copies data from the EEPROM to the EXTRACTBUFF Note that this buffer is only 31 bytes long although this routine can support up to 256 bytes. |

|  |  |
| --- | --- |
| Routine | **SAVE\_EXTRACTBUF** - $515D/$514C |
| Parameters | PARM\_LEN - Number of bytes to be copied INST\_ADDRHI:INST\_ADDRLO - Address in Prom to write EXTRACTBUF - Contains the bytes to write to the EEPROM |
| Purpose | This copies data from the EXTRACTBUFF to the EEPROM Note that this buffer is only 31 bytes long although this routine can support up to 256 bytes. |

|  |  |
| --- | --- |
| Routine | **SYSTEM\_RESET** - $519B/$518A |
| Parameters | None |
| Purpose | This routine is the main reset routine for starting up the watch. It cleans up all of memory and starts the processing once again |

|  |  |
| --- | --- |
| Routine | **INIT\_SOUNDS** - $5265/$51F2 |
| Parameters | None |
| Purpose | This routine initializes the default sounds |

|  |  |
| --- | --- |
| Routine | **ENABLE\_EYE** - $53A6/$5367 |
| Parameters | None |
| Purpose | This routine enables the received on the watch to download from the screen. It also seems to wait for SERIAL\_DATA/SERIAL\_CONTROL to settle down |

|  |  |
| --- | --- |
| Routine | **DISABLE\_EYE** - $53BD/$537E |
| Parameters | None |
| Purpose | This disables the eye for normal watch operation |

|  |  |
| --- | --- |
| Routine | **SET\_SYS\_07** - $53C8/$5389 |
| Parameters | None |
| Purpose | ???? This routine resets the SYS\_07 hardware |

|  |  |
| --- | --- |
| Routine | **CLEAR\_SYS\_07** - $53CF/$5390 |
| Parameters | None |
| Purpose | ???? This routine resets the SYS\_07 hardware Clears 1,HW\_FLAGS |

|  |  |
| --- | --- |
| Routine | **RESET\_SYS\_07** - $53D5/$5396 |
| Parameters | None |
| Purpose | ???? This routine resets the SYS\_07 hardware |

|  |  |
| --- | --- |
| Routine | **INITHW\_SYS\_07** - $53DC/$539D |
| Parameters | SYSTEMP2 - 0 or $c1 to indicate how the hardware is to be reset |
| Purpose | ???? This routine initializes the SYS\_07 hardware |

|  |  |
| --- | --- |
| Routine | **SETHW\_07\_08\_C1** - $53F4/$53B5 |
| Parameters | A - $C1 - Value to be poked into SYS\_08 |
| Purpose | ???? Resets the SYS\_07, SYS\_08 hardware. There is a timing loop associated with this reset operation. |

|  |  |
| --- | --- |
| Routine | **WRITE\_ACQUIRE** - $543C/$542B |
| Parameters | None |
| Purpose | This routine acquires the EEPROM for writing. It will also turn off any playing sound as well as the INDIGLO in order to conserve power while doing the writing. |

|  |  |
| --- | --- |
| Routine | **WRITE\_RELEASE** - $5448/$5437 |
| Parameters | None |
| Purpose | This routine releases the EEPROM for writing. If the Indiglo had been previously on, it is turned back on. |

|  |  |
| --- | --- |
| Routine | **MAKE\_INST2\_LDA\_X** - $5453/$5442 **MAKE\_INST2\_STA\_X** - $5457/$5446 |
| Parameters | None |
| Purpose | These routines make the INST2 opcodes to be an LDA or STA $nnnn,X instruction |

|  |  |
| --- | --- |
| Routine | **PROM\_READ** - $5462/$5451 |
| Parameters | INST2\_COUNT - Number of bytes to be copied PROM\_ADDRHI:PROM\_ADDRLO - Address in Prom to read INST2\_ADDRHI:INST2\_ADDRLO - Address to copy data to |
| Purpose | This copies data from the EEPROM to the indicated buffer |

|  |  |
| --- | --- |
| Routine | **PROM\_WRITE** - $5488/$5477 |
| Parameters | INST2\_COUNT - Number of bytes to be copied PROM\_ADDRHI:PROM\_ADDRLO - Address in Prom to write INST2\_ADDRHI:INST2\_ADDRLO - Address to copy data from |
| Purpose | This copies data to the EEPROM from the indicated buffer |

|  |  |
| --- | --- |
| Routine | **SET\_INDIGLO** - $5504/$54F3 |
| Parameters | 0,HW\_FLAGS – Indicates request for on or off |
| Purpose | This routine turns on/off the Indiglo light |

### Scrolling Messages

|  |  |
| --- | --- |
| Routine | **PUTSCROLLMSG** - $5522/$5511 |
| Parameters | MSGBUF - the message to scroll terminated by SEPARATOR |
| Purpose | Initialize a scrolling message |

|  |  |
| --- | --- |
| Routine | **SCROLLMSG** - $5545/$5534 |
| Parameters | MSGBUF - Message to be scroll terminated by a SEPARATOR character |
| Purpose | Start the scrolling cycle for the current message |

|  |  |
| --- | --- |
| Routine | **SCROLLMSG\_CONT** - $5549/$5538 |
| Parameters | MSGBUF - Message to be scroll terminated by a SEPARATOR character SCROLL\_TICS - The current tic count in the cycle |
| Purpose | Start the scrolling cycle for the current message, but don’t reset the scrolling cycle wait count. |

### Blinking routines

|  |  |  |  |
| --- | --- | --- | --- |
| Routine | **START\_BLINKX** - $55BB/$55AA | | |
| Parameters | A - Blinking function to be selected | | |
| 0 | BLINK\_YEAR | Blink the year in the right place according to the current time format |
| 1 | BLINK\_SECONDS | Blink two characters point to by UPDATE\_PARM on the right two digits of the middle line - Used for the seconds |
| 2 | BLINK\_AMPM | Blink AM/PM on the right most digits of the middle line (A or P pointed to by UPDATE\_PARM) |
| 3 | BLINK\_MONTH | Blink the month in the right place according to the current time format |
| 4 | BLINK\_HMONTH | Blink the month in the right place according to the current time format for a half date (no year) |
| 5 | BLINK\_DAY | Blink the day in the right place according to the current time format |
| 6 | BLINK\_HDAY | Blink the day in the right place according to the current time format for half dates |
| 7 | BLINK\_MID12 | Blink the left two blank padded digits on the middle line (value pointed to by UPDATE\_PARM) |
| 8 | BLINK\_HOUR | Blink the Hour (left two segments on the middle line) and AM/PM indicator (hour point to by UPDATE\_PARM) |
| 9 | BLINK\_MID34 | Blink the middle two zero padded digits on the middle line (value pointed to by UPDATE\_PARM) |
| 10 | BLINK\_SEGMENT | Blink a single segment indicated by UPDATE\_POS and mask in UPDATE\_VAL |
| 11 | BLINK\_DIGIT | Blink solid black cursor for the digit (UPDATE\_POS is the location on the bottom line) |
| 12 | BLINK\_TZONE | Blink the timezone information (Pointed to by UPDATE\_PARM) |
| 13 | BLINK\_TOP34 | Blink the middle zero padded two digits on the top line (value pointed to by UPDATE\_PARM) |
| X - single byte parameter to the particular blinking function | | |
| Purpose | Establish and call the specified blinking routine | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Routine | **START\_BLINKP** - $55BF/$55AE | | |
| Parameters | A - Blinking function to be selected | | |
| 0 | BLINK\_YEAR | Blink the year in the right place according to the current time format |
| 1 | BLINK\_SECONDS | Blink two characters point to by UPDATE\_PARM on the right two digits of the middle line - Used for the seconds |
| 2 | BLINK\_AMPM | Blink AM/PM on the right most digits of the middle line (A or P pointed to by UPDATE\_PARM) |
| 3 | BLINK\_MONTH | Blink the month in the right place according to the current time format |
| 4 | BLINK\_HMONTH | Blink the month in the right place according to the current time format for a half date (no year) |
| 5 | BLINK\_DAY | Blink the day in the right place according to the current time format |
| 6 | BLINK\_HDAY | Blink the day in the right place according to the current time format for half dates |
| 7 | BLINK\_MID12 | Blink the left two blank padded digits on the middle line (value pointed to by UPDATE\_PARM) |
| 8 | BLINK\_HOUR | Blink the Hour (left two segments on the middle line) and AM/PM indicator (hour point to by UPDATE\_PARM) |
| 9 | BLINK\_MID34 | Blink the middle two zero padded digits on the middle line (value pointed to by UPDATE\_PARM) |
| 10 | BLINK\_SEGMENT | Blink a single segment indicated by UPDATE\_POS and mask in UPDATE\_VAL |
| 11 | BLINK\_DIGIT | Blink solid black cursor for the digit (UPDATE\_POS is the location on the bottom line) |
| 12 | BLINK\_TZONE | Blink the timezone information (Pointed to by UPDATE\_PARM) |
| 13 | BLINK\_TOP34 | Blink the middle zero padded two digits on the top line (value pointed to by UPDATE\_PARM) |
| X - Address of parameter to the particular blinking function | | |
| Purpose | Establish and call the specified blinking routine | | |

### Update functions

|  |  |  |  |
| --- | --- | --- | --- |
| Routine | **START\_UPDATEX** - $57C3/$56C4 | | |
| Parameters | A - Update function to be selected | | |
| 0 | UPD\_YEAR | Update the year |
| 1 | UPD\_MONTH | Update the Month |
| 2 | UPD\_HMONTH | Update the Month in Half date format |
| 3 | UPD\_DAY | Update the day |
| 4 | UPD\_HDAY | Update the day in half date format |
| 5 | UPD\_MID12 | Update MID12 |
| 6 | UPD\_HOUR | Update the hour |
| 7 | UPD\_MID34 | Update MID34 |
| 8 | UPD\_DIGIT | Update the digit at UPDATE\_POS |
| X - single byte parameter to the particular update function | | |
| Purpose | Establish and call the specified update function | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Routine | **START\_UPDATEP** - $57C7/$56C8 | | |
| Parameters | A - Update function to be selected | | |
| 0 | UPD\_YEAR | Update the year |
| 1 | UPD\_MONTH | Update the Month |
| 2 | UPD\_HMONTH | Update the Month in Half date format |
| 3 | UPD\_DAY | Update the day |
| 4 | UPD\_HDAY | Update the day in half date format |
| 5 | UPD\_MID12 | Update MID12 |
| 6 | UPD\_HOUR | Update the hour |
| 7 | UPD\_MID34 | Update MID34 |
| 8 | UPD\_DIGIT | Update the digit at UPDATE\_POS |
| X - Pointer to parameters for the update function | | |
| Purpose | This establishes an update function. Update functions are called every 8/10th of a second. This function will update a number in an upward or downward direction based on the setting of 0,SYSFLAGS | | |

### Format Routines

These routines are useful for formatting numbers into the corresponding character representation.

|  |  |
| --- | --- |
| Routine | **FMTXLEAD0** - $593E/$583F |
| Parameters | X - value to be formatted. 0-9 results in 0 followed by the digit 10-99 results in number for both digits |
| Purpose | Formats into DATDIGIT1/2 with leading zeros |

|  |  |
| --- | --- |
| Routine | **FMTBLANK0** - $594D/$584E |
| Parameters | X - value to be formatted. 0 results in all blanks. 1-9 results in blank followed by the digit 10-99 results in number for both digits |
| Purpose | Formats a number into DATDIGIT1/2 |

|  |  |
| --- | --- |
| Routine | **FMTX** - $5951/$5852 |
| Parameters | X - value to be formatted. 0-9 results in blank followed by the digit 10-99 results in number for both digits |
| Purpose | Formats a number into DATDIGIT1/2 |

|  |  |
| --- | --- |
| Routine | **FMTSPACE** - $595C/$585D |
| Parameters | None |
| Purpose | This routine simply puts spaces into DATDIGIT1 DATDIGIT2 |

|  |  |
| --- | --- |
| Routine | **FMTBLANK0B** - $5963/$5864 |
| Parameters | X - value to be formatted. 0 results in all blanks. 1-9 results in blank followed by the digit 10-99 results in number for both digits |
| Purpose | Formats a number into DATDIGIT1/2. This routine does not appear to be used anywhere and seems to do exactly the same thing as FMTBLANK0 |

|  |  |
| --- | --- |
| Routine | **FIXLEAD0** - $5A2A/$592B |
| Parameters | None |
| Purpose | If the first digit is a zero, replace it with a blank |

### Line routines

These routines are useful for putting strings on the display

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Routine | **PUTLINE3** - $56D5/$59E7 | | | | | | | | |
| Parameters | A = Position | | | | | | | |  |
| S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 |  |
| $47 | $3D | $33 | $27 | $1D | $13 | $09 | $0a |
| X = Character in Timex ASCII to display | | | | | | | |  |
| Purpose | Put a single character on the bottom line of the display This routine pokes in a single digit on the display. Note that the last digit is backwards and upside down in the hardware. | | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Routine | **PUTLINE1** - $570D/$5A33 | | | | | | |
| Parameters | A = Position | | | | | |  |
| T1 | T2 | T3 | T4 | T5 | T6 |  |
| $46 | $3E | $34 | $2C | $22 | $14 |
| X = Character in Timex ASCII to display | | | | | |  |
| Purpose | Put a single character on the top line of the display | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Routine | **PUTLINE2** - $5745/$5A6B | | | | | | |
| Parameters | A = Position | | | | | |  |
| M1 | M2 | M3 | M4 | M5 | M6 |  |
| $46 | $3E | $34 | $2C | $22 | $14 |
| X = Character in Timex ASCII to display | | | | | |  |
| Purpose | Put a single character on the second line of the display | | | | | | |

|  |  |
| --- | --- |
| Routine | **SETALL** - $5776/$5A9C |
| Parameters | None |
| Purpose | Turns on all segments on the entire display |

|  |  |
| --- | --- |
| Routine | **CLEARALL** - $577A/$5AA0 |
| Parameters | None |
| Purpose | Clear the entire display |

|  |  |
| --- | --- |
| Routine | **CLEARBOT** - $5787/$5AAD |
| Parameters | None |
| Purpose | Clear the bottom line of the display |

|  |  |
| --- | --- |
| Routine | **CLEAR\_RANGE** - $5793/$5ABF |
| Parameters | A – Initial offset to be clearing from X – Number of words to clear |
| Purpose | Turn off all bits on the display at the given offsets |

|  |  |
| --- | --- |
| Routine | **CLEARSYM** - $579F/$5ACB |
| Parameters | None |
| Purpose | Turns off all the non digit symbols segments (including dots, dashes and colons) |

|  |  |
| --- | --- |
| Routine | **BANNER8** - $5845/$5746 |
| Parameters | A = Offset from 0110 for the start of an 8 character Timex string |
| Purpose | Display an 8 character string |

|  |  |
| --- | --- |
| Routine | **PUTMSGXBOT** - $5849/$574A |
| Parameters | A = Message selector number. Valid values from 0 to 27. They correspond to the same strings passed into PUTMSGBOT scaled down by 8 |
| Purpose | Display an 8 character system string on the bottom line |

|  |  |  |  |
| --- | --- | --- | --- |
| Routine | **PUTMSGBOT** - $584C/$574D | | |
| Parameters | A = Offset into message selector string. Valid values from $00 to $d8 at 8 Byte offsets. $E0 is the start of the 6 byte top/mid message strings. | | |
| $00 | SYS8\_MON | “MON     “ |
| $08 | SYS8\_TUE | “TUE     “ |
| $10 | SYS8\_WED | “WED     “ |
| $18 | SYS8\_THU | “THU     “ |
| $20 | SYS8\_FRI | “FRI     “ |
| $28 | SYS8\_SAT | “SAT     “ |
| $30 | SYS8\_SUN | “SUN     “ |
| $38 | SYS8\_VERDATE | “ 802003 “ |
| $40 | SYS8\_VERSION | “  V2.0  “ |
| $48 | SYS8\_MODE | “  MODE  “ |
| $50 | SYS8\_SET\_MODE | “SET MODE” |
| $58 | SYS8\_SET | “SET     “ |
| $60 | SYS8\_TO | “TO      “ |
| $68 | SYS8\_FOR | “FOR     “ |
| $70 | SYS8\_ENTRIES | “ENTRIES “ |
| $78 | SYS8\_UPCOMING | “UPCOMING” |
| $80 | SYS8\_ENTRY | “ ENTRY  “ |
| $88 | SYS8\_SCAN | “ SCAN   “ |
| $90 | SYS8\_SCAN\_RIGHT | “    SCAN” |
| $98 | SYS8\_SYNCING | “ SYNCING” |
| $a0 | SYS8\_PROGRESS | “PROGRESS” |
| $a8 | SYS8\_DATA\_OK | “ DATA OK” |
| $b0 | SYS8\_RESEND | “-RESEND-“ |
| $b8 | SYS8\_ABORTED | “ ABORTED” |
| $c0 | SYS8\_MISMATCH | “MISMATCH” |
| $c8 | SYS8\_SPLIT | “ SPLIT  “ |
| $d0 | SYS8\_START | “>=START “ |
| $d8 | SYS8\_STOP | “>=STOP  “ |
| Purpose | Display an 8 character system string on the bottom line | | |

|  |  |
| --- | --- |
| Routine | **PUTDOWTOP** - $5872/$5773 |
| Parameters | X - Day of week (0-6) |
| Purpose | Displays the two character representation of the day of the week in the upper left of the display |

|  |  |
| --- | --- |
| Routine | **PUT6TOP** - $587E/$577F |
| Parameters | A = Offset from WRIST\_MAIN for the start of a 6 byte data item to be put on the top line of the screen. This uses a different encoding for characters where: we have 32 different values which correspond to: 0123456789ABCDEFGH:LMNPRTUWYr -+ e.g. $12=’:’, $13=’L’. It appears that things wrap when you get to $20 |
| Purpose | Display a 6 character string on the top line |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine | **PUTMSG1** - $5882/$5783 | | | |
| Parameters | A = Offset into message selector string. Valid values from $00 to $a8 at 6 Byte offsets. | | | |
| $00 | SYS6\_SET | “ SET  “ | |
| $06 | SYS6\_HOLDTO | “HOLDTO” | |
| $0C | SYS6\_ALARM | “ALARM “ | |
| $12 | SYS6\_ENTER | “ENTER “ | |
| $18 | SYS6\_HR | “    HR” | |
| $1E | SYS6\_SWITCH | “SWITCH” | |
| $24 | SYS6\_TIME | “ TIME “ | |
| $2A | SYS6\_FORMAT | “FORMAT” | |
| $30 | SYS6\_DAILY | “DAILY “ | |
| $36 | SYS6\_APPT | “ APPT “ | |
| $3c | SYS6\_NO | “  NO  “ | |
| $42 | SYS6\_APPTS | “APPTS “ | |
| $48 | SYS6\_END\_OF | “END OF” | |
| $4e | SYS6\_LIST | “ LIST “ | |
| $54 | SYS6\_DELETE | “DELETE” | |
| $5a | SYS6\_ANN | “ ANN  “ | |
| $60 | SYS6\_PHONE | “PHONE “ | |
| $66 | SYS6\_DONE | “ DONE “ | |
| $6c | SYS6\_PRI | “PRI   “ | |
| $72 | SYS6\_COMM | “ COMM “ | |
| $78 | SYS6\_READY | “READY “ | |
| $7e | SYS6\_IN | “  IN  “ | |
| $84 | SYS6\_ERROR | “ERROR “ | |
| $8a | SYS6\_CEASED | “CEASED” | |
| $90 | SYS6\_PC | “PC-   “ | |
| $96 | SYS6\_WATCH | “WATCH “ | |
| $9c | SYS6\_CHRONO | “CHRONO” | |
| $A2 | SYS6\_TIMER | “TIMER “ | |
| $a8 | SYS6\_000000 | “000000” | |
| Purpose | Display an 6 character system string on the top line | | |

|  |  |
| --- | --- |
| Routine | **PUT6MID** - $58A8/$57A9 |
| Parameters | A = Offset from WRIST\_MAIN for the start of a 6 byte data item to be put on the top line of the screen. This uses a different encoding for characters where: we have 32 different values which correspond to: 0123456789ABCDEFGH:LMNPRTUWYr -+ e.g. $12=’:’, $13=’L’. Beyond $20 you get random junk. |
| Purpose | Display a 6 character string on the second line |

|  |  |
| --- | --- |
| Routine | **PUTMSG2** - $58AC/$57AD |
| Parameters | A = Offset into message selector string. Valid values from $00 to $a8 at 6 Byte offsets and the strings are the same as for [PUTMSG1](#PUTMSG1) |
| Purpose | Display an 6 character system string on the top line |

|  |  |
| --- | --- |
| Routine | **CLEARTOP** - $58D2/$57D3 |
| Parameters | None |
| Purpose | Puts blanks into all 6 top digits (Blanks out the top line) |

|  |  |
| --- | --- |
| Routine | **CLEARMID** - $58D8/$57D9 |
| Parameters | None |
| Purpose | Puts blanks into all 6 Middle digits (Blanks out the middle line) |

|  |  |
| --- | --- |
| Routine | **CLRTOP12** - $58DE/$57DF |
| Parameters | None |
| Purpose | Puts blanks into top Digits 1 and 2 |

|  |  |
| --- | --- |
| Routine | **PUTTOP12** - $58E0/$57E1 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into TOP Digits 1 and 2 |

|  |  |
| --- | --- |
| Routine | **CLRTOP34** - $58EE/$57EF |
| Parameters | None |
| Purpose | Puts blanks into TOP Digits 3 and 4 |

|  |  |
| --- | --- |
| Routine | **PUTTOP34** - $58F0/$57F1 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into TOP Digits 3 and 4 |

|  |  |
| --- | --- |
| Routine | **CLRTOP56** - $58FE/$57FF |
| Parameters | None |
| Purpose | Puts blanks into TOP Digits 5 and 6 |

|  |  |
| --- | --- |
| Routine | **PUTTOP56** - $5900/$5801 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into TOP Digits 5 and 6 |

|  |  |
| --- | --- |
| Routine | **CLRMID12** - $590E/$580F |
| Parameters | None |
| Purpose | Puts blanks into Middle Digits 1 and 2 |

|  |  |
| --- | --- |
| Routine | **PUTMID12** - $5910/$5811 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into Middle Digits 1 and 2 |

|  |  |
| --- | --- |
| Routine | **CLRMID34** - $591E/$581F |
| Parameters | None |
| Purpose | Puts blanks into Middle Digits 3 and 4 |

|  |  |
| --- | --- |
| Routine | **PUTMID34** - $5920/$5821 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into Middle Digits 3 and 4 |

|  |  |
| --- | --- |
| Routine | **CLRMID56** - $592E/$582F |
| Parameters | None |
| Purpose | Puts blanks into Middle digits 5 and 6 |

|  |  |
| --- | --- |
| Routine | **PUTMID56** - $5930/$5831 |
| Parameters | None |
| Purpose | Puts DATDIGIT1/2 into Middle Digits 5 and 6 |

|  |  |
| --- | --- |
| Routine | **SAYEOLMSG** - $5979/$587A |
| Parameters | None |
| Purpose | Puts ‘END OF LIST’ on the display |

|  |  |
| --- | --- |
| Routine | **SAYHOLDTODELETE** - $598A/$588B |
| Parameters | None |
| Purpose | Puts ‘HOLD TO DELETE ENTRY’ on the display |

|  |  |
| --- | --- |
| Routine | **PUT\_PHONENUM** - $59A2/$58A3 |
| Parameters | None |
| Purpose | Puts a phone number on the top two lines of the display (Up to 12 digits). If there is a non blank character as the third digit, a - is turned on between the 3rd and 4th digits to separate out what is presumably the area code |

|  |  |
| --- | --- |
| Routine | **PUTYEARMID** - $59D9/$58DA |
| Parameters | X - Year to be formatted on the display |
| Purpose | Puts the current year on the right half of the middle display. If the year passed in is less than 50, it is assumed to be 20xx, above 50 it is processed as 19xx giving a range of 1950-2049 |

|  |  |
| --- | --- |
| Routine | **CLEAR\_HMONTH** - $59F8/$58F9 |
| Parameters | None |
| Purpose | blank out the 2 character day for a half date (no year) based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUT\_HMONTHX** - $59FD/$58FE |
| Parameters | X - Day to be displayed |
| Purpose | Put the leading space 2 digit month in the appropriate spot on the display based on the current time zone date format for a half date (no year) |

|  |  |
| --- | --- |
| Routine | **CLEAR\_HDAY** - $5A11/$5912 |
| Parameters | None |
| Purpose | blank out the 2 character day for a half date (no year) based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUT\_HDAYX** - $5A16/$5917 |
| Parameters | X - Day to be displayed |
| Purpose | Put the leading zero 2 digit day in the appropriate spot on the display based on the current time zone date format for a half date (no year) |

|  |  |
| --- | --- |
| Routine | **CLEAR\_MONTH** - $5A36/$5937 |
| Parameters | None |
| Purpose | blank out the 2 character month based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **CLEAR\_DAY** - $5A4F/$5950 |
| Parameters | None |
| Purpose | blank out the 2 character day based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUTBOT678** - $5A86/$5987 |
| Parameters | X - Pointer to 3 byte location containing bytes to put on the display (pointed to by x) 3 bytes in TIMEX ASCII. Because the X register is used to index to them, they must be located in the first 256 bytes of memory. |
| Purpose | Puts three digits into the lower corner of the display. Typically this is the time zone information. |

|  |  |
| --- | --- |
| Routine | **CLEAR\_YEAR** - $5A6F/$5970 |
| Parameters | None |
| Purpose | blank out the 2 character year based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **IPUT\_MONTHX** - $5A3B/$593C |
| Parameters | X - Month to be displayed |
| Purpose | Put the leading space 2 digit month in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **IPUT\_DAYX** - $5A54/$5955 |
| Parameters | X - Day to be displayed |
| Purpose | Put the leading zero 2 digit day in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **IPUT\_YEARX** - $5A74/$5975 |
| Parameters | X - Year to be displayed |
| Purpose | Put the leading zero 2 digit year in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUTHALFDATESEP** - $5AA0/$59A1 |
| Parameters | None |
| Purpose | Show the separator character for a half date (no year) based on the current date format |

|  |  |
| --- | --- |
| Routine | **PUTDATESEP** - $5AAB/$59AC |
| Parameters | None |
| Purpose | Show the separator characters for a full date based on the current date format |

|  |  |
| --- | --- |
| Routine | **PUT\_LETTERX** - $5ACE/$59CF |
| Parameters | A - Character to be displayed X - Offset on the bottom line to put character |
| Purpose | Put a single character at the appropriate spot on the bottom line |

|  |  |
| --- | --- |
| Routine | **PUT\_HOURX** - $5AD9/$59DA |
| Parameters | X - Hour to be displayed |
| Purpose | Put the hour on the first two digits of the middle line along with the colon |

|  |  |
| --- | --- |
| Routine | **UPDATE\_SECONDS** - $625E/$6267 |
| Parameters | None |
| Purpose | This routine checks the current TIC count and updates the seconds based on that TIC. If the minute rolls over, we also set the flags so that the rest of the system can respond to it. |

|  |  |
| --- | --- |
| Routine | **SHOW\_TIME\_DISPLAY** - $676A/$6773 |
| Parameters | None |
| Purpose | Display the time information based on the current time zone and whether or not we might be in time set mode. All symbols are updated |

|  |  |
| --- | --- |
| Routine | **PUT\_YEARX** - $67CC/$67D5 |
| Parameters | X - Year to be displayed |
| Purpose | Put the leading zero 2 digit year in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUT\_MONTHX** - $67D0/$67D9 |
| Parameters | X - Month to be displayed |
| Purpose | Put the leading space 2 digit month in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **PUT\_DAYX** - $67D4/$67DD |
| Parameters | X - Day to be displayed |
| Purpose | Put the leading zero 2 digit day in the appropriate spot on the display based on the current time zone date format |

|  |  |
| --- | --- |
| Routine | **SAY\_HOURX** - $67D8/$67E1 |
| Parameters | X - Hour to be displayed |
| Purpose | Puts up the hour on the display along with an AM/PM indicator and a Colon. This code respects the current 12/24 hour format. |

|  |  |
| --- | --- |
| Routine | **CLEAR\_PM** - $6815/$681C |
| Parameters | NONE |
| Purpose | Turn off the PM indicator. |

|  |  |
| --- | --- |
| Routine | **CLEAR\_AM** - $681C/$6825 |
| Parameters | NONE |
| Purpose | Turn off the AM indicator. |

|  |  |
| --- | --- |
| Routine | **PUT\_MINUTEX** - $6823/$682C |
| Parameters | X - minute (0-59) to be displayed |
| Purpose | This puts the minute in the middle two digits on the middle line followed by a period |

|  |  |
| --- | --- |
| Routine | **SHOWSEC\_TENS** - $6830/$6839 |
| Parameters | SECOND\_TENS - Value to be put on the display |
| Purpose | Puts the character at SECOND\_TENS onto the next to the last digit on the middle line |

|  |  |
| --- | --- |
| Routine | **SHOWSEC\_ONES** - $6838/$6841 |
| Parameters | SECOND\_ONES – Value to be put on the display |
| Purpose | Puts the character at SECOND\_ONES onto the last digit on the middle line |

|  |  |
| --- | --- |
| Routine | **SHOWNIGHT\_SYM** - $6840/$6849 |
| Parameters | None |
| Purpose | Displays the night symbol if we are in night mode |

|  |  |
| --- | --- |
| Routine | **SAY\_HOLD\_TO** - $6855/$685E |
| Parameters | None |
| Purpose | Puts ‘HOLD-TO’ on the top line |

|  |  |
| --- | --- |
| Routine | **FIX\_TMAPP\_DAY** - $6861/$686A |
| Parameters | None |
| Returns | A - limited day of the month |
| Purpose | Based on TMAPP\_MONTH, TMAPP\_YEAR, this routine limits the day of the month to a legal one |

|  |  |
| --- | --- |
| Routine | **TMAPP\_COPYTZ1** - $6881/$688A |
| Parameters | None |
| Purpose | Copies the Hour, Minute, Month, Day, and Year information for Time Zone 1 to the corresponding TMAPP variables. |

|  |  |
| --- | --- |
| Routine | **TMAPP\_COPYTZ2** - $688C/$6895 |
| Parameters | None |
| Purpose | Copies the Hour, Minute, Month, Day, and Year information for Time Zone 2 to the corresponding TMAPP variables. |

|  |  |
| --- | --- |
| Routine | **GETTZNAME** - $6897/$68A0 |
| Parameters | None |
| Returns | X - Pointer to the 3 character name of the current time zone |

|  |  |
| --- | --- |
| Routine | **GET\_MONTHDAYX** - $689F/$68A8 |
| Parameters | X - pointer to two byte location to retrieve Month and Day |
| Returns | A - The current year for the current time zone |
| Purpose | Returns the year for the current time zone |

|  |  |
| --- | --- |
| Routine | **GET\_YEAR** - $68B2/$68BB |
| Parameters | None |
| Returns | A - The current year for the current time zone |
| Purpose | Returns the year for the current time zone |

|  |  |
| --- | --- |
| Routine | **GET\_HOURFORMAT** - $68BB/$68C4 |
| Parameters | None |
| Returns | X - 12 or 24 depending on the time format |
| Purpose | Returns the 12/24 hour time format for the current time zone |

|  |  |
| --- | --- |
| Routine | **GET\_DATEFMT** - $68CB/$68D4 |
| Parameters | None |
| Returns | A - Date format mask for the current time zone One of: 0 = DATEFMT\_MMDDYY = Date Format is MM-DD-YY 1 = DATEFMT\_DDMMYY = Date Format is DD-MM-YY 2 = DATEFMT\_YYMMDD = Date Format is YY-MM-DD and One of 0 = DATEFMT\_SEPDASH = Dates are separated by dashes 4 = DATEFMT\_SEPDOTS = Dates are separated by periods |
| Purpose | Returns the date format for the current time zone |

|  |  |
| --- | --- |
| Routine | **CALC\_DOW\_X** - $68D5/$68DE |
| Parameters | X - Pointer to Month, Day, Year block |
| Purpose | Computes the Day of the Week from the Month, Day, Year information |

|  |  |
| --- | --- |
| Routine | **COPY\_MDY** - $68DB/$68E4 |
| Parameters | X - pointer to Month, Day, Year block to copy |
| Purpose | Copies over the Month, Day, and Year information in preparation for calling CALC\_DOW |

|  |  |
| --- | --- |
| Routine | **ACQUIRE** - $68E8/$68F1 |
| Parameters | None |
| Purpose | Disable interrupts for a short piece of code |

|  |  |
| --- | --- |
| Routine | **RELEASE** - $68F2/$68FB |
| Parameters | None |
| Purpose | Reenable interrupts |

|  |  |
| --- | --- |
| Routine | **GET\_MONTHLEN** - $68F9/$6902 |
| Parameters | PARM\_MONTH, PARM\_YEAR contain the month and year to look for |
| Returns | A - Number of days in the month |
| Purpose | Computes the number of days in a given month |

|  |  |
| --- | --- |
| Routine | **CHECK\_TZ** - $690E/$6917 |
| Parameters | None |
| Purpose | Determine which time zone is to be displayed. Carry flag clear = TZ1 Carry flag set = TZ2 |

|  |  |
| --- | --- |
| Routine | **CALC\_DOW** - $691C/$6925 |
| Parameters | CURRENT\_MONTH, CURRENT\_DAY, CURRENT\_YEAR - holds the information to calculate from |
| Returns | A - Day of Week (0=Monday...6=Sunday) |
| Purpose | Calculates the day of the week from the given information |

|  |  |
| --- | --- |
| Routine | **LIST\_DISPLAY\_CURRENT** - $6ABB/$6AC4 |
| Parameters | None |
| Purpose | Display the current list entry. List entries are up to 31 bytes long with Byte 0: Completion status. Negative numbers indicate that it is not yet done Byte 1: The priority of the event. 0 indicates no priority Bytes 2-26: The packed text of the message (Up to 32 bytes unpacked) Bytes 27-31 – Wasted since they can never be unpacked |

|  |  |
| --- | --- |
| Routine | **INCA\_WRAPX** - $6B0D/$6B16 |
| Parameters | A - Number to be incremented X - Range to hold number within |
| Purpose | Advance to the next value wrapped within a range |

|  |  |
| --- | --- |
| Routine | **DELAY\_X** - $6B31/$6B3A |
| Parameters | X - Delay interval (Measured in ?) - Note that 1 is the only value ever passed in here |
| Purpose | Delay for a fixed amount of time |

|  |  |
| --- | --- |
| Routine | **DELAY\_X16** - $6B43/$6B4C |
| Parameters | X - interval to delay for ($C8 is the only value ever passed in) |
| Purpose | Delay for a fixed amount of time |

|  |  |
| --- | --- |
| Routine | **GETBCDHI** - $6B52/$6B5B |
| Parameters | X - Hex value to be converted (Range 0-99) |
| Returns | A - High byte of number in Timex ASCII |

|  |  |
| --- | --- |
| Routine | **GETBCDLOW** - $6B5A/$6B63 |
| Parameters | X - Hex value to be converted (Range 0-99) |
| Returns | A - Low byte of number in Timex ASCII |

|  |  |
| --- | --- |
| Routine | **ALARM\_CHECK** - $6BC4/$6C9C |
| Parameters | None |
| Purpose | This routine is called once a minute to check for and raise any alarms |

|  |  |
| --- | --- |
| Routine | **SHOWNOTE\_SYM** - $6C62/$6C56 |
| Parameters | None |
| Purpose | Displays the NOTE symbol if there is a note to be displayed |

|  |  |
| --- | --- |
| Routine | **SHOWALARM\_SYM** - $6C76/$6C6A |
| Parameters | None |
| Purpose | Displays the ALARM symbol if there are any enabled alarms which are not masked This will also start the alarm symbol blinking if we are in alarm backup mode |

|  |  |
| --- | --- |
| Routine | **ALARM\_DISPLAY\_CURRENT** - $6EF4/$6EFD |
| Parameters | None |
| Purpose | Display the current alarm information on the entire display. Daily is put on the top line and the NOTE/ALARM symbols are displayed accordingly |

|  |  |
| --- | --- |
| Routine | **ALARM\_SHOW\_HOURLYNOTE** - $6F39/$6F42 |
| Parameters | None |
| Purpose | Set the note symbol to the state of the hourly chimes |

|  |  |
| --- | --- |
| Routine | **ALARM\_SHOW\_ALARMSYM** - $6F4A/$6F53 |
| Parameters | ALARM\_FLAGS – status of alarm to show |
| Purpose | Set the alarm symbol to the state of the current alarm |

|  |  |
| --- | --- |
| Routine | **ALARM\_SHOW\_AMPM** - $6F5B/$6F64 |
| Parameters | ALARM\_FLAGS - indicates whether a 12 hour format is in AM or PM |
| Purpose | Set the alarm symbol to the state of the current alarm |

|  |  |
| --- | --- |
| Routine | **MASK\_ALARMS** - $6FF3/$6FFC |
| Parameters | None |
| Purpose | This temporarily disables all alarms by turning on the mask bit (0x02) for all five alarms. |

|  |  |
| --- | --- |
| Routine | **UNMASK\_ALARMS** - $7000/$7009 |
| Parameters | None |
| Purpose | This reenables all alarms by turning off the mask bit (0x02) for all five alarms. |

|  |  |
| --- | --- |
| Routine | **ANNIV\_SHOW\_DATE** - $7184/$718D |
| Parameters | None |
| Purpose | Displays date for the current anniversary entry |

|  |  |
| --- | --- |
| Routine | **ANNIV\_SHOW\_SCAN\_DATE** - $719F/$71A8 |
| Parameters | None |
| Purpose | Displays date for the current anniversary scan date |

|  |  |
| --- | --- |
| Routine | **ANNIV\_SHOW\_CURRENT** - $71AC/$71B5 |
| Parameters | None |
| Purpose | Displays the current anniversary entry |

|  |  |
| --- | --- |
| Routine | **SHOWREMIND\_SYM** - $71D6/$71DF |
| Parameters | None |
| Purpose | Displays the reminder symbol if there are any anniversaries within this week. If one is today, this will toggle the remind symbol each time this routine is called |

|  |  |
| --- | --- |
| Routine | **OFFREMIND\_SYM** - $71EE/$71F7 |
| Parameters | None |
| Purpose | Turns off the reminder symbol |

|  |  |
| --- | --- |
| Routine | **SAY\_NO\_ANN\_ENTRIES** - $71F5/$71FE |
| Parameters | None |
| Purpose | Displays the message NO ANN ENTRIES on the display |

|  |  |
| --- | --- |
| Routine | **APPT\_SHOW\_TIME** - $73D7/$73E0 |
| Parameters | SCAN\_QHOUR – the quarter hour to display |
| Purpose | This shows the appointment time on the display (including AM/PM indicator) |

|  |  |
| --- | --- |
| Routine | **APPT\_SHOW\_DATE** - $7439/$7442 |
| Parameters | SCAN\_MONTH,SCAN\_DAY |
| Purpose | This shows the appointment date on the display (including the day of the week) |

|  |  |
| --- | --- |
| Routine | **APPT\_SHOW\_SCAN** - $7454/$745D |
| Parameters | SCAN\_MONTH,SCAN\_DAY |
| Purpose | This shows the scan date on the display (including the day of the week) with the year and a message indicating that we are scanning |

|  |  |
| --- | --- |
| Routine | **APPT\_SHOW\_CURRENT** - $7461/$746A |
| Parameters | None |
| Purpose | This shows the next upcoming appointment (if any) |

|  |  |
| --- | --- |
| Routine | **APPT\_SHOW\_UPCOMING** - $748E/$7497 |
| Parameters | None |
| Purpose | This shows the next upcoming appointment (if any) |

|  |  |
| --- | --- |
| Routine | **SAY\_NO\_APPT\_ENTRIES** - $74BD/$74C6 |
| Parameters | None |
| Purpose | This puts NO APPT ENTRIES on the display |

|  |  |
| --- | --- |
| Routine | **COMM\_CHECK\_CRC** - $7C56/$7C3C |
| Parameters | None |
| Returns | A - 0 CRC for the current packet matched $ff - CRC for the current packet did not match |
| Purpose | Compute and validate a CRC for the current packet |

## Installing a Wristapp

Many people have asked how to install a Wristapp and download it to your watch.  While there are people who are using their DataLink with many different operating systems, these instructions only work for the Timex Data Link software for Windows (what comes on the floppy disk with the watch).  Note that this is different than Schedule+ or another PIM downloading to the watch.

1. Locate the directory where the DataLink software is installed.  Typically this will be C:\Datalink or C:\Program Files\DataLink.  In that directory will be a file called TimexDL.DAT
2. Using your favorite editor (Notepad will work just fine), bring in that file to edit.
3. Search in the file for the [WristApps] section.  It will consist of several lines like:

[WristApps]

WristAppTotal=10

SelectedWristApp=9

WristAppSendOption=True

WristApp000=HEXDUMP0.ZAP

WristApp001=Melody17.ZAP

WristApp002=HELLO.ZAP

WristApp003=NUMBER.ZAP

WristApp004=Update.ZAP

WristApp005=Flash.ZAP

WristApp006=passwd.ZAP

WristApp007=dayfind.ZAP

WristApp008=testsnd.ZAP

WristApp009=endoff.ZAP

1. Note the number in the WristAppTotal and increment it by one.  (In this case I would change the 10 to an 11)
2. Go to the last entry and add a new line just like the ones above it, but increment the WristApp number by one.  In this case, I would add a line after the WristApp009=endoff.ZAP and call that line WristApp010=.  Put the name of the wristapp (don’t forget the .ZAP extension) on the line.  In my example, it would look like:

[WristApps]

WristAppTotal=11

SelectedWristApp=9

WristAppSendOption=True

WristApp000=HEXDUMP0.ZAP

WristApp001=Melody17.ZAP

WristApp002=HELLO.ZAP

WristApp003=NUMBER.ZAP

WristApp004=Update.ZAP

WristApp005=Flash.ZAP

WristApp006=passwd.ZAP

WristApp007=dayfind.ZAP

WristApp008=testsnd.ZAP

WristApp009=endoff.ZAP

WristAPP010=NewApp.ZAP

1. Save the file
2. Copy the .ZAP file into the APP subdirectory of the DataLink software and you are done.
3. Load up the Datalink Software, and click on the WristApps button.
4. Scroll to the bottom of the list to see your new WristApp
5. Select the wristapp and make sure that the bottom says to send the selected WristApp
6. Select OK and then proceed to download to your watch with the normal COMM mode
7. Enjoy!

# My Wristapps

The wristapps that I have written so far.  Everything here works for both the 150 and the 150s.

1. [*TipCalc*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\tipcalc.zip) - Calculates 10, 15, 20% tips. Thanks to David M. Schreck <[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)> for the idea!
2. [*Hello*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hello.zip) - [Tutorial #1](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hello.html) - Hello World! (Now where is my C Compiler?)
3. [*Number*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\number.zip) - [Tutorial #2](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\number.html) - Change a single number
4. [*Update*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\update.zip) - [Tutorial #3](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\update.html) - Update a number using a system routine
5. [*Flash*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\flash.zip) - [Tutorial #4](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\flash.html) - Blinks and changes the number.
6. [*Passwd*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\passwd.zip) - [Tutorial #5](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\passwd.html) - Blinks, changes, and selects numbers.
7. [*DayFind*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\dayfind.zip) - [Tutorial #6](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\dayfind.html) - gives you the day of the week
8. [*Sound Test*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\testsnd.zip) - [Tutorial #7](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\testsnd.html) - Plays one of the 14 possible tones on the watch.
9. [*EndOff*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\endoff.zip) - [Tutorial #8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\endoff.html) - Turn off alarms on the weekend
10. [*HexDump*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hexdump.zip) - [Tutorial #9](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hexdump.html) - Dump out memory.
11. [*PromDump*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\promdump.zip) - [Tutorial #10](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\promdump.html) - Dump out the contents of the EEPROM.
12. [*SpendWatch*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\spend0.zip) - [Tutorial #11](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\spend0.html) - Track how much you spend in a day.
13. [*Sound1*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\sound1.zip) - [Tutorial #12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\sound1.html) - Create a simple soundscape.
14. [*3Ball*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\3ball.zip) - [Tutorial #13](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\3ball.html) - Can’t make up your mind?  Let 3Ball help you out.  Thanks to *Wayne Buttles <*[timex@fdisk.com](mailto:timex@fdisk.com)*>*
15. [*ShipBell*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\shipbell.zip) - [Tutorial #14](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\shipbell.html) - Beeps on the hour with the number of hours past a shift change. (suggested by *“Theron E. White, CPA” <*[twhite@mercury.peganet.com](mailto:twhite@mercury.peganet.com)*>*).
16. [*Data Hider*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\datahid2.zip) - This works for both the 150 and the 150s.
17. Segment Setter - This allows you to set all of the segments on the display on/off.

## Other People’s Wristapps

It is wonderful to now see other people creating Wristapps.

1. [*NumPad*](http://www.panix.com/~mpoly/software/numpad.zip) - Michael Polymenakos <[mpoly@panix.com](mailto:mpoly@panix.com)> has created an excellent app which has two functions in one.  In his own words: “The first thing I miss from my old (and now non-functional) Casio is the ability to record a number quickly when pen and paper are not available. I wrote a small wristapp, NUMPAD, to let me record a 12 digit number... Any comments will be appreciated (especially on replacing the ugly cursor with a ‘blink’ function that blinks only one digit at a time).”  He has also incorporated a chronometer wristapp in with the app to give you two apps in one.
2. [*3Ball*](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\3ball.zip) - Wayne Buttles <[timex@fdisk.com](mailto:timex@fdisk.com)> created the first version of this fun app.  It’s been updated here as a tutorial.

## Plans for Wristapps

The wristapps that I plan to create and know everything necessary to create them.

1. WestMinister Chimes - For that ‘Big Ben’ sound.  With thanks to Pigeon for the sound scheme to make it possible.

Other wristapps that have been suggested (their original comments are presented. I also include  my comments in blue).

1. Falling Blocks - I have been thinking about this for a while.  There is really no reason that you can’t design a game to take advantage of the segments to do a simple falling-blocks-like game.  You would have to turn the display sideways to play it.
2. Slots - I have also wanted to do this game for a while.  The basic idea is to have a slot machine in the watch where you can press a button and take a whirl.  The watch should keep track of your winnings.  Because of the way the segments are organized, I believe that you can even do a good imitation of the wheels spinning.
3. Dumper - We need to have a good application that allows the Datalink to talk back to the PC.  The obvious way here will be to use the sounds on the watch and listen to them with the SoundBlaster on the PC.  Right now the only thing holding us back is someone to create the PC end to listen.  I have everything necessary to generate the tones in a predictable manner.
4. Phone Dialer - The Datalink is just screaming for this application that has been suggested by many people.  It is not clear that this is beyond the capabilities of the DataLink, but so far I have only been able to emit the 14 basic tones in the watch.  From my understanding of the watch and the hardware, I haven’t completely ruled this out as a possibility.
5. Info entry - “One of the reasons I like the DataLink is because it DOESN’T have an ugly 12 button keypad on it, but I have to admit, it would be nice to be able to enter a phone number when needed. Granted, it would cumbersome to enumerate the desired digits, but I think it would still be useful (could also be used to enter the section # of a large parking lot that you left your car).*” David M. Schreck <*[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)*>*.  This is certainly doable, but it does have some issues to be considered in dealing with the EEProm. See the [**EEProms**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\eeprom.html) information to understand why.
6. Screen Saver - “Not in the true sense of the phrase, of course, and this one you would have to purposely invoke. I imagine that those who are artistically inclined might think up a creative and interesting way to cycle through the available display fields.” *David M. Schreck <*[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)*>*.  If someone proposes a suggested way that this might work, I certainly could implement it.
7. Baseball counter - “This might be too simple to bother with, but people who are umpires (I’m mainly thinking about the many folks who ump for little league games) use a little hand held clicker to keep track of balls, strikes, and outs. This should be an easy applet to create.” *David M. Schreck <*[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)*>*  This is one where I would love to hear from someone who would actually use it.  I have a number of ideas for user interface, but that would really depend on how someone would use it.
8. Tennis counter - “Say I’m about to start a tennis game. I hit one button each time I score a point, and a different one each time my opponent scores. The applet always displays the current score. It might even display the word “deuce” when appropriate. Hopefully it could be programmed to be smart enough to know when subsequent games begin, and even keep track of the set score.” *David M. Schreck <*[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)*>*.  Here is where I will let my lack of knowledge of tennis show.  I simply don’t know how the scoring works well enough to write this.  I would like to have the person enter the two names of the people playing and it would keep track of who has to serve, the current score, and the total match/set score.  If someone would toss me this information, I could create the app really quickly.
9. Calorie Counter - “If someone wanted to keep track of their caloric intake for the day (or any other need where you want to tally up a total but don’t feel like carrying around a paper and pencil) perhaps they could just punch in the number to be added to the daily total each time they eat something. At the end of the day they can glance at the total and then reset to zero. *David M. Schreck <*[dschreck@csfbg.csfb.com](mailto:dschreck@csfbg.csfb.com)*>*” This is probably one of the more interesting apps to create.  I might even take advantage of the EEProm to store some of the basic foods and their calorie counts to make it easier.

# Wristapp Programming Tutorial

## A First Wristapp - Hello World

To illustrate, let us take our favorite C Program and figure out how to put it on the Datalink.  The first step in creating a wristapp is to decide on what the user interface will be.  You would think that with only 5 buttons, this would be an easy task, but in reality this can make or break a good application. For our application, we will have it so that when you first enter the app, it puts “HELLO WORLD MODE” on the screen.  If you press the PREV button, it will toggle to turning on all segments.  Pressing the PREV button will switch back to the “HELLO WORLD MODE”.  The Next button will take you out of the app and the SET/NEXT buttons will not do anything.  Pressing the GLOW button will activate the Indiglo light as expected.  Here’s what the code would look like:

;Name: Hello World

;Version: HELLO

;Description: This is a simple Hello Program

;by John A. Toebes, VIII

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 0 indicates that we want to show the segments instead of the message

;

START EQU \*

;

; (2) System entry point vectors

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

S6\_HELLO: timex6 “HELLO “

S6\_WORLD: timex6 “WORLD “

;

; (4) State Table

; (4) State Table

STATETAB:

db 0

db EVT\_ENTER,TIM\_ONCE,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_DNNEXT,TIM\_ONCE,0 ; Next button

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events. We only see ENTER, RESUME, and DNNEXT events

;

HANDLE\_STATE0:

bset 1,$8f ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_DNNEXT ; Did they press the next button?

beq DOTOGGLE ; Yes, toggle what we are displaying

CLEARIT bclr 0,FLAGBYTE ; Start us in the show display state

REFRESH brclr 0,FLAGBYTE,SHOWDISP ; Do we want to see the main display?

jmp [SETALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#SETALL) ; No, just turn on all segments

SHOWDISP jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_HELLO-START ; Get the offset for the first string

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP) ; And send it to the top line

lda #S6\_WORLD-START ; Get the offset for the second string

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID) ; and put it on the middle line

lda #SYS8\_MODE ; Get the system offset for the ‘MODE’ string

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT) ; and put it on the bottom line

;

; (6) Our only real piece of working code...

DOTOGGLE brset 0,FLAGBYTE,CLEARIT ; If it is set, just jump to clear it like normal

bset 0,FLAGBYTE ; Already clear, so set it

bra REFRESH ; and let the refresh code handle it

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta $96

clr FLAGBYTE ; start with a clean slate

rts

Now all of that code needs a little explanation.  As you can see from the numbers, we have 7 basic sections

1. [Program specific constants](#Hello_1) - This is where you declare everything that you want to use.  As a Wristapp, you have only a limited amount of Ram (7 bytes to be specific) that you can store your stuff with, so be careful here.
2. [System entry point vectors](#Hello_2) - These are fixed and mandated for any Wristapp.  If there is more than one state, the JMP and db sequence is repeated for each state.
3. [Program strings](#Hello_3) - In order to provide addressability to the strings, you need to put them immediately after the entry point vectors.
4. [State Table(s)](#Hello_4) - This really tells the watch how we want to operate and what events we want to handle.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.
5. [State Table Handler(s)](#Hello_5) - These are called to process the events for a particular state.  Typically this is a LDA  BTNSTATE followed by a lot of CMP/Bcc instructions.  You also need to do the BSET 1,$8f at the start to allow the Wristapp to be suspendable.
6. [Program Specific Code](#Hello_6) - The actual meat of the program.  In our case, we simply have to toggle a value.
7. [Main Initialization routine](#Hello_7) - This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).

Now that we have a basic program working.  Next Up:  [**Getting Input - Numbers**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\numbers.html)

## Getting Input

A program which just does output and really takes no input is not very useful.  The first stage in making a program more useful is to figure out how to allow the user to enter a value.  With this first numbers program, we allow you to enter a number by pressing the PREV/NEXT key to advance it by one each time you press the key.   This allows us to see how basic input works and a couple of the formatting/display routines.

;Name: Numbers

;Version: NUMBER

;Description: This is a simple number count program

;by John A. Toebes, VIII

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 0 indicates that we want to show the segments instead of the message

;

CURVAL EQU $62 ; The current value we are displaying

START EQU \*

;

; (2) System entry point vectors

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

S6\_NUMBER: timex6 “NUMBER”

S6\_COUNT: timex6 “COUNT “

;

; (4) State Table

STATETAB:

db 0

db EVT\_ENTER,TIM2\_8TIC,0 ; Initial state

db EVT\_TIMER2,TIM\_ONCE,0 ; The timer from the enter event

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_DNNEXT,TIM\_ONCE,0 ; Next button

db EVT\_DNPREV,TIM\_ONCE,0 ; Prev button

db EVT\_DNSET,TIM\_ONCE,0 ; Set button

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events. We will see ENTER, RESUME, DNNEXT, DNPREV, DNSET, and TIMER2

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_DNNEXT ; Did they press the next button?

beq DO\_NEXT ; Yes, increment the counter

cmp #EVT\_DNPREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_DNSET ; Maybe the set button?

beq DO\_SET ; Deal with it!

cmp #EVT\_ENTER ; Is this our initial entry?

bne REFRESH

;

; This is the initial event for starting us

;

DO\_ENTER

bclr 1,FLAGBYTE ; Indicate that we need to clear the display

jsr [CLEARSYM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARSYM) ; Clear the display

lda #S6\_NUMBER-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_COUNT-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #SYS8\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

;

; (6) Our only real working code...

DO\_NEXT

inc CURVAL

lda CURVAL

cmp #100

bne SHOWVAL

DO\_SET

clr CURVAL

SHOWVAL

brset 1,FLAGBYTE,NOCLEAR

REFRESH

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL)

bset 1,FLAGBYTE

NOCLEAR

ldx CURVAL

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0)

jmp [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34)

DO\_PREV

lda CURVAL

beq WRAPUP

dec CURVAL

bra SHOWVAL

WRAPUP

lda #99

sta CURVAL

bra SHOWVAL

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr FLAGBYTE ; start with a clean slate

clr CURVAL

rts

We have the same 7 basic sections, but some of them are a little more filled out.

1. [Program specific constants](#Numbers_1) - We have only two basic variables.  The flagbyte and the current value.
2. [System entry point vectors](#Numbers_2) - We have nothing special this time..
3. [Program strings](#Numbers_3) - The strings go here for addressability.
4. [State Table(s)](#Numbers_4) - This really tells the watch how we want to operate and what events we want to handle.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this. For this, we want to see the down events for the NEXT, PREV, and SET buttons so that we can increment, decrement, or reset the counter as appropriate.  We also have coded the MODE button with the magic $FF which causes it to advance to the next app.
5. [State Table Handler(s)](#Numbers_5) - Here we have the typical CMP/BEQ instruction sequence to quickly determine what event happened.  Note that the EVT\_ENTER event causes a timer to go off which allows us to clear the screen 8/10 second after they switch to the app.
6. [Program Specific Code](#Numbers_6) - The actual meat of the program.  We really only have to deal with advance/retreat/reset of the value and then displaying it after each change..
7. [Main Initialization routine](#Numbers_7) - This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).

Just pressing a button for each increment can be tedious.  Learn how to make it better with: [**Better Input - Update**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\update.html)

## Better Input - Update

Pressing the button for each time you want to increment or decrement a number can be very tedious.  Fortunately, the Datalink has a series of update routines that you can call to handle this automatically.  The update routine takes a few parameters.  First is the type of update to do.  The function limits

:

;Name: Update

;Version: UPDATE

;Description: This is a simple number update program

;by John A. Toebes, VIII

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 1 indicates that we need to clear the display first

;

CURVAL EQU $62 ; The current value we are displaying

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

S6\_UPDATE: timex6 “UPDATE”

S6\_SAMPLE: timex6 “SAMPLE”

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM\_2\_8TIC,0 ; Initial state

db EVT\_TIMER2,TIM\_ONCE,0 ; The timer from the enter event

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_DNANY4,TIM\_ONCE,0 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,0 ; NEXT, PREV, SET, MODE button released

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, RESUME, DNANY4 and UPANY4 events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_DNANY4 ; Did they press a button?

bne CHKENTER ; No, pass on to see what else there might be

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_SET ; Perhaps the SET button

beq DO\_SET ; If so, handle it

; In reality, we can’t reach here since we handled all three buttons

; in the above code (the MODE button is handled before we get here and the

; GLOW button doesn’t send in an event for this). We can just fall through

; and take whatever we get from it.

CHKENTER

cmp #EVT\_ENTER ; Is this our initial entry?

bne REFRESH

;

; This is the initial event for starting us

;

DO\_ENTER

bclr 1,FLAGBYTE ; Indicate that we need to clear the display

jsr [CLEARSYM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARSYM) ; Clear the display

lda #S6\_UPDATE-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_SAMPLE-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #SYS8\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

;

; (6) Our real working code...

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

clra

sta UPDATE\_MIN ; Our low end is 0

lda #99

sta UPDATE\_MAX ; and the high end is 99 (the max since this is a 2 digit value)

ldx #CURVAL ; Point to our value to be updated

lda #UPD\_MID34 ; Request updating in the middle of the display

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP) ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

bclr 1,FLAGBYTE

lda #SYS8\_SET\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

DO\_SET

clr CURVAL ; When they hit the set button, we just clear to zero

SHOWVAL

brset 1,FLAGBYTE,NOCLEAR ; Do we need to clear the display first?

REFRESH

jsr CLEARALL ; Yes, clear everything before we start

bset 1,FLAGBYTE ; And remember that we have already done that

NOCLEAR

bclr 7,BTNFLAGS ; Turn off any update routine that might be pending

ldx CURVAL ; Get the current value

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0) ; Convert it to the two ASCII digits

jmp [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34) ; And put it on the screen in the right place

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr FLAGBYTE ; start with a clean slate

clr CURVAL

rts

Now all of that code needs a little explanation.  As you can see from the numbers, we have 7 basic sections

1. [Program specific constants](#update_1) - This is where you declare everything that you want to use.  As a Wristapp, you have only a limited amount of Ram (7 bytes to be specific) that you can store your stuff with, so be careful here.
2. [System entry point vectors](#update_2) - These are fixed and mandated for any Wristapp.  If there is more than one state, the JMP and db sequence is repeated for each state.  We haven’t started getting fancy so we still have only one state table.
3. [Program strings](#update_3) - In order to provide addressability to the strings, you need to put them immediately after the entry point vectors.  Our only strings are the two banner strings.
4. [State Table(s)](#update_4) - This really tells the watch how we want to operate and what events we want to handle.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.  We accept the normal RESUME, ENTER, and TIMER2 events for getting us running.  We also handle the MODE button by allowing it to just bounce us out of the application and into the next.  It is important that this event be in the table before the EVT\_DNANY4 which allows for the NEXT, PREV, SET, and MODE buttons (it ignores the INDIGLO button).  If you press the mode button, it will be handled by the first entry and the application terminated cleanly.  Otherwise, we have to sort out which of the three buttons was pressed.  This is easy to do since BTN\_PRESSED holds the actual code associated with the button that was selected.
5. [State Table Handler(s)](#update_5) - These are called to process the events for a particular state.  Typically this is a LDA  BTNSTATE followed by a lot of CMP/Bcc instructions.  You also need to do the BSET 1,$8f at the start to allow the Wristapp to be suspendable.  In this case we introduce the use of the EVT\_DNANY4 in the basic state table logic testing.  When we see the EVT\_DNANY4 or EVT\_UPANY4, we look at BTN\_PRESSED to identify what the user pressed.
6. [Program Specific Code](#update_6) - The actual meat of the program.  Again, the code is very simple.  We have to handle making sure that the screen is cleared at the appropriate times, but other than that, the majority of the work is picking a direction and setting 0.SYSFLAGS appropriately before letting the system handle the Update for us.  Once we are set up, we set 4,BTNFLAGS and the system roms will handle updating the number for us.
7. [Main Initialization routine](#update_7) - This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).

This has gotten a bit better for input, now you need to show them what they have selected with: [**Showing Selection - Blink**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\blink.html)

## Showing Selection - Blink routines

We can make our update program a bit smarter and more obvious to the user by blinking the digit when it is available to be changed.  Like the START\_UPDATEP routine, there is an equivalent START\_BLINKP routine which handles blinking the display for you.  I call this routine FLASH since it is not possible to put a K on the top two lines of the display :-).

;Name: Flash

;Version: FLASH

;Description: by John A. Toebes, VIII

;This is a simple number update/flash program

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 1 indicates that we need to clear the display first

;

CURVAL EQU $62 ; The current value we are displaying

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

S6\_FLASH: timex6 “FLASH “

S6\_SAMPLE: timex6 “SAMPLE”

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM\_2\_8TIC,0 ; Initial state

db EVT\_TIMER2,TIM\_ONCE,0 ; The timer from the enter event

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_DNANY4,TIM\_ONCE,0 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,0 ; NEXT, PREV, SET, MODE button released

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, RESUME, DNANY4 and UPANY4 events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_DNANY4 ; Did they press a button?

bne CHKENTER ; No, pass on to see what else there might be

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_SET ; Perhaps the SET button

beq DO\_SET ; If so, handle it

; In reality, we can’t reach here since we handled all three buttons

; in the above code (the MODE button is handled before we get here and the

; GLOW button doesn’t send in an event for this). We can just fall through

; and take whatever we get from it.

CHKENTER

cmp #EVT\_ENTER ; Is this our initial entry?

bne REFRESH

;

; This is the initial event for starting us

;

DO\_ENTER

bclr 1,FLAGBYTE ; Indicate that we need to clear the display

jsr [CLEARSYM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARSYM) ; Clear the display

lda #S6\_FLASH-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_SAMPLE-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #SYS8\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

;

; (6) Our real working code...

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

clra

sta UPDATE\_MIN ; Our low end is 0

lda #99

sta UPDATE\_MAX ; and the high end is 99 (the max since this is a 2 digit value)

ldx #CURVAL ; Point to our value to be updated

lda #UPD\_MID34 ; Request updating in the middle of the display

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP) ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

bclr 1,FLAGBYTE

lda #SYS8\_SET\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

DO\_SET

clr CURVAL ; When they hit the set button, we just clear to zero

SHOWVAL

brset 1,FLAGBYTE,NOCLEAR ; Do we need to clear the display first?

REFRESH

jsr CLEARALL ; Yes, clear everything before we start

bset 1,FLAGBYTE ; And remember that we have already done that

NOCLEAR

bclr 7,BTNFLAGS ; Turn off any update routine that might be pending

ldx #CURVAL

lda #BLINK\_MID34

jsr START\_BLINKP

bset 2,BTNFLAGS ; Mark a blink routine as pending

rts

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr FLAGBYTE ; start with a clean slate

clr CURVAL

rts

This is code is basically identical to the [**Update**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\update.html) sample with only a couple of minor changes.

1. [Program specific constants](#flash_1) - No Change.
2. [System entry point vectors](#flash_2) - We have nothing special this time..
3. [Program strings](#flash_3) - Gee, we changed the strings.
4. [State Table(s)](#flash_4) - We get to use exactly the same state table.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.
5. [State Table Handler(s)](#flash_5) - Since the state table is the same, the state handling is the same.
6. [Program Specific Code](#flash_6) - All we had to do different here was to call [START\_BLINKP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_BLINKP) and then set 2,BTNFLAGS to notify the system that we want the blink routine to run.  The blink routine will automatically handle putting up the number for us.
7. [Main Initialization routine](#flash_7) - No changes here either. This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).

## Entering Digits - PASSWD sample

This program is a bit more sophisticated to show off how you might go toward creating a complex app.  I have not made any attempts at optimizing the code here in order to be a bit more clear about how to go about writing this type of app.  There are a few new features with this code:

1. We have two different display screens.  When you first enter the app, it puts up one display.  After it times out, it puts up a different display which also has a scrolling message across the bottom.
2. The set button brings you into a set mode where the mode button switches between digits to set.
3. This app uses two state tables instead of one.  It shows how to switch between the two states.

;Name: Password

;Version: PASSWD

;Description: This is a simple number update/passwd program

;by John A. Toebes, VIII

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 0 indicates which digit we are working on (SET=SECOND DIGIT)

; Bit 1 indicates that we need to clear the display first

;

DIGIT0 EQU $62 ; The first digit to enter

DIGIT1 EQU $63 ; The second digit to enter

SYSTEMP0 EQU $A0

SYSTEMP1 EQU $A1

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB0-STATETAB0

L0127: jmp HANDLE\_STATE1

db STATETAB1-STATETAB0

;

; (3) Program strings

S6\_TOEBES: timex6 “TOEBES”

S6\_SAMPLE: timex6 “SAMPLE”

S6\_PRESS: timex6 “PRESS “

S8\_PASSWORD: Timex “PASSWORD”

SX\_MESSAGE Timex “BY JOHN A. TOEBES, VIII”

db SEPARATOR

;

; (4) State Table

;

STATETAB0:

db 0

db EVT\_ENTER,TIM\_2\_8TIC,0 ; Initial state

db EVT\_TIMER2,TIM\_ONCE,0 ; The timer from the enter event

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_SET,TIM\_ONCE,1 ; SET button pressed

db EVT\_END

STATETAB1:

db 1

db EVT\_RESUME,TIM\_ONCE,1 ; Resume from a nested app

db EVT\_DNANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button released

db EVT\_USER2,TIM\_ONCE,0

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_ENTER ; Is this our initial entry?

bne REFRESH0

;

; This is the initial event for starting us

;

DO\_ENTER

bclr 1,FLAGBYTE ; Indicate that we need to clear the display

jsr CLEARSYM ; Clear the display

lda #S6\_TOEBES-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_SAMPLE-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #S8\_PASSWORD

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

;

; We come here for a RESUME or TIMER2 event. For this we want to reset the display

;

REFRESH0

brset 1,FLAGBYTE,NOCLEAR0 ; Do we need to clear the display first?

bset 1,FLAGBYTE

jsr [CLEARSYM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARSYM)

NOCLEAR0

lda #S6\_PRESS-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #SYS6\_SET

jsr [PUTMSG2](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSG2)

lda #SX\_MESSAGE-START

jmp [SETUP\_SCROLL](#SETUP_SCROLL)

;

; (6) State Table 1 Handler

; This is called to process the state events.

; We see SET, RESUME, DNANY4, and UPANY4 events

;

HANDLE\_STATE1:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_UPANY4

beq REFRESH

cmp #EVT\_DNANY4 ; Is this our initial entry?

bne FORCEFRESH

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_MODE ; Perhaps the MODE button

beq DO\_MODE ; If so, handle it

; It must be the set button, so take us out of this state

lda #EVT\_USER2

jmp [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT)

;

; (7) Our real working code...

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

clra

sta UPDATE\_MIN ; Our low end is 0

lda #99

sta UPDATE\_MAX ; and the high end is 99 (the max since this is a 2 digit value)

brset 0,FLAGBYTE,UPD1

ldx DIGIT1

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0)

jsr [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34)

ldx #DIGIT0 ; Point to our value to be updated

lda #UPD\_MID12 ; Request updating in the middle of the display

bra UPD2

UPD1

ldx DIGIT0

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0)

jsr [PUTMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID12)

ldx #DIGIT1

lda #UPD\_MID34

UPD2

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP) ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

bclr 1,FLAGBYTE

lda #SYS8\_SET\_MODE

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

DO\_MODE

lda FLAGBYTE

eor #1

sta FLAGBYTE

REFRESH

brset 1,FLAGBYTE,NOCLEAR ; Do we need to clear the display first?

FORCEFRESH

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Yes, clear everything before we start

bset 1,FLAGBYTE ; And remember that we have already done that

NOCLEAR

bclr 7,BTNFLAGS ; Turn off any update routine that might be pending

brset 0,FLAGBYTE,SET1

ldx DIGIT1

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0)

jsr [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34)

ldx #DIGIT0

lda #BLINK\_MID12

bra SET2

SET1

ldx DIGIT0

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0)

jsr [PUTMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID12)

ldx #DIGIT1

lda #BLINK\_MID34

SET2

jsr [START\_BLINKP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_BLINKP)

bset 2,BTNFLAGS ; Mark a blink routine as pending

rts

;

; (8) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr FLAGBYTE ; start with a clean slate

clr DIGIT0

clr DIGIT1

rts

;

; (9) This subroutine is useful for getting a scrolling string on the screen

;

;----------------------------------------------------------------------

; Routine:

; SETUP\_SCROLL

; Parameters:

; X - Offset from Start to the string

; Returns:

; MSGBUF - contains copied string

; Purpose

; This copies the current string into MSGBUF and calls the appropriate routines

; to start it scrolling on the bottom line.

;----------------------------------------------------------------------

SETUP\_SCROLL:

clr SYSTEMP0

sta SYSTEMP1

DO\_COPY:

ldx SYSTEMP1 ; Get the pointer to the source character

lda START,X ; Get the character that we are copying

ldx SYSTEMP0 ; Get the pointer to the output buffer

sta MSGBUF,X ; and store the character away

inc SYSTEMP0 ; Increment our count

inc SYSTEMP1 ; As well as the pointer to the character

cmp #SEPARATOR ; Did we get a terminator character

bne DO\_COPY ; No, go back for more

;

; The string is now in a buffer terminated by a separator character

;

jsr [PUTSCROLLMSG](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTSCROLLMSG) ; Initialize the scrolling support

jmp [SCROLLMSG](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#SCROLLMSG) ; And tell it to actually start scrolling

This is code is built on the [Update](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\update.html) and [Blink](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\blink.html) samples with a few changes and additions.

1. [Program specific constants](#passwd_1) - We now have two digits to care about.
2. [System entry point vectors](#passwd_2) - Because we have gone to two state tables, we now have the extra jump vector.
3. [Program strings](#passwd_3) - Gee, we changed the strings.  Plus we have a longer string which we pass to our [SETUP\_SCROLL](#SETUP_SCROLL) routine.
4. [State Table(s)](#passwd_4) - We now have two state tables.  State table0 is pretty simple and is used only for when we are in the normal state.  State table 1 is used when we are in the set mode.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.
5. [State Table Handler0](#passwd_5) - For state0, we only really need to handle the initial enter where we put up the banner.  After a while we time out and put up the ‘PRESS SET’ message with my name scrolling across the bottom.
6. [State Table Handler1](#passwd_6) - This handler is used for when we are in the SET state for changing the numbers.
7. [Program Specific Code](#passwd_6) - We use the same UPDATE and BLINK functions from the [Blink](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\blink.html) sample.  The only extra work here is that we cause the display to update the other digit when we are setting one.
8. [Main Initialization routine](#passwd_7) - No changes here. This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).
9. [SETUP\_SCROLL subroutine](#passwd_9) - This is a useful routine that you may wish to copy for another wristapp.

## Getting time and Input - DAYFIND sample

This is the first real app with some attempt at optimization and a bit of planning for user input.  It stems from a suggestion by Roman Mazi.  There are a lot of things in this code which build on the previous examples.  The most notable things in this one are:

1. This code shows how to get the current date (and you can also get the time the same way).
2. There are banner messages on the bottom of the display to provide a little help.
3. Workarounds for a lack of update routines are given.
4. Quite a few new routines are introduced here.

The code is reasonably commented:

;Name: Day Finder

;Version: DAYFIND

;Description: This will allow you to determine the date for a given day of the week and vice-versa.

;by John A. Toebes, VIII

;

;Press the prev/next buttons to advance by a single day. Press SET to access the ability to advance/backup by

;weeks, months, days, and years. The MODE button advances through those different states

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

B\_CLEAR EQU 0 ; Bit 0 indicates that we need to clear the display first

B\_SCANUP EQU 1 ; Bit 1 indicates that we are scanning up

B\_SCANNING EQU 2 ; Bit 2 indicates that we are in a fake scanning mode

DIGSEL EQU $62 ; Indicates which digit we are working on

; 0 = DAY OF WEEK

; 1 = Month

; 2 = Day

; 3 = Year

YEAR\_DIG1 EQU $63 ; This is the first digit of the year to blink (the tens digit)

YEAR\_DIG2 EQU $64 ; This is the second digit of the year to blink (the ones digit)

COUNTER EQU $65 ; A convenient counter for us to advance a week at a time

;

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB0-STATETAB0

L0127: jmp HANDLE\_STATE1

db STATETAB1-STATETAB0

;

; (3) Program strings

S6\_DAY timex6 “DAY “

S6\_FIND timex6 “ FIND”

S8\_TOEBES Timex “J.TOEBES”

S8\_DAYFIND Timex “DAY FIND”

S8\_WEEK db C\_LEFTARR

Timex “ WEEK “

db C\_RIGHTARR

S8\_MONTH db C\_LEFTARR

Timex “MONTH “

db C\_RIGHTARR

S8\_DAY db C\_LEFTARR

Timex “ DAY “

db C\_RIGHTARR

S8\_YEAR db C\_LEFTARR

Timex “ YEAR “

db C\_RIGHTARR

;

; (4) State Table

;

STATETAB0:

db 0

db EVT\_ENTER,TIM1\_4TIC,0 ; Initial state

db EVT\_TIMER1,TIM\_ONCE,0 ; The timer from the enter event

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_SET,TIM\_ONCE,1 ; SET button pressed

db EVT\_DNNEXT,TIM2\_8TIC,0 ; NEXT button pressed

db EVT\_DNPREV,TIM2\_8TIC,0 ; PREV button pressed

db EVT\_UPANY4,TIM\_ONCE,0 ; The

db EVT\_TIMER2,TIM2\_TIC,0 ; The timer for the next/prev button pressed

db EVT\_END

STATETAB1:

db 1

db EVT\_RESUME,TIM\_ONCE,1 ; Resume from a nested app

db EVT\_DNANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button released

db EVT\_USER2,TIM\_ONCE,0

db EVT\_USER3,TIM2\_8TIC,1 ;

db EVT\_TIMER2,TIM2\_TIC,1 ;

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_DNNEXT

beq DO\_NEXT0

cmp #EVT\_DNPREV

beq DO\_PREV0

cmp #EVT\_TIMER2

beq DO\_SCAN

cmp #EVT\_ENTER ; Is this our initial entry?

bne REFRESH0

;

; This is the initial event for starting us up

;

DO\_ENTER

;

; (6) This code gets the current date from the system

jsr [ACQUIRE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#ACQUIRE) ; Lock so that it doesn’t change under us

ldx #TZ1\_MONTH ; Assume that we are using the first timezone

jsr [CHECK\_TZ](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CHECK_TZ) ; See which one we are really using

bcc COPY\_TZ1 ; If we were right, just skip on to do the work

ldx #TZ2\_MONTH ; Wrong guess, just load up the second time zone

COPY\_TZ1

lda 0,x ; Copy out the month

sta SCAN\_MONTH

lda 1,x ; Day

sta SCAN\_DAY

lda 2,x ; and year

sta SCAN\_YEAR

jsr [RELEASE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#RELEASE) ; Unlock so the rest of the system is happy

bclr B\_CLEAR,FLAGBYTE ; Indicate that we need to clear the display

clr DIGSEL ; Start us off on the week advance

jsr [CLEARSYM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARSYM) ; Clear the display

lda #S6\_DAY-START

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_FIND-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #S8\_TOEBES-START

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

DO\_SCAN

brclr B\_SCANUP,FLAGBYTE,DO\_PREV0 ; Were we scanning up or down?

DO\_NEXT0

bset B\_SCANUP,FLAGBYTE ; We are now scanning up

jsr [INCREMENT\_SCAN\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#INCREMENT_SCAN_DATE) ; Advance to the next date

bra SHOW\_DATE ; Comment this out and use the next one if you want

; jmp [APPT\_SHOW\_SCAN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#APPT_SHOW_SCAN) ; to put the text ‘SCAN’ on the bottom when we are in scan mode

DO\_PREV0

bclr B\_SCANUP,FLAGBYTE ; We are now scanning down

jsr [DECREMENT\_SCAN\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#DECREMENT_SCAN_DATE) ; Back up to the previous date

bra SHOW\_DATE ; Show the date on the screen.

; jmp [APPT\_SHOW\_SCAN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#APPT_SHOW_SCAN) ; Use this if you want ‘SCAN’ on the bottom of the display

;

; We come here for a RESUME or TIMER2 event. For this we want to reset the display

;

REFRESH0

brset B\_CLEAR,FLAGBYTE,NOCLEAR0 ; Do we need to clear the display first?

bset B\_CLEAR,FLAGBYTE ; Mark that the display has been cleared

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; and do the work of clearing

NOCLEAR0

lda #S8\_DAYFIND-START ; Put up the name of the app on the display

jsr [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

SHOW\_DATE

jsr [APPT\_SHOW\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#APPT_SHOW_DATE) ; Show the date on the screen

ldx SCAN\_YEAR ; as well as the year

jmp [PUTYEARMID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTYEARMID)

;--------------------------------------------------------------------------------

; (7) State Table 1 Handler

; This is called to process the state events.

; We see SET, RESUME, USER3, TIMER2, DNANY4, and UPANY4 events

; We use the USER3 to trigger a delay which fires off a TIMER2 sequence of events.

; This allows us to have the PREV/NEXT buttons repeat for advancing the WEEK and YEAR

; since we can’t use the UPDATE routines for them.

;

HANDLE\_STATE1:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_TIMER2 ; Was it a timer for a repeat operation?

beq DO\_UPD ; Yes, go handle it

cmp #EVT\_USER3 ; Was it the USER3 event fired from the PREV/NEXT buttons?

bne TRY\_UP ; No, try again

rts ; Yes, just ignore it, it will cause a timer to go off later

TRY\_UP

bclr B\_SCANNING,FLAGBYTE ; We can’t be scanning any more, so turn it off

cmp #EVT\_UPANY4 ; Was it any button being released?

bne TRY\_DN ; No, try again

jmp REFRESH ; Yes, go refresh the screen (note that the branch is out of range)

TRY\_DN

cmp #EVT\_DNANY4 ; Is this our initial entry?

beq GET\_DN ; No, try again

jmp FORCEFRESH ; Yes, go setup the screen (note that the branch is out of range)

GET\_DN

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_MODE ; Perhaps the MODE button

beq DO\_MODE ; If so, handle it

; It must be the set button, so take us out of this state

lda #EVT\_USER2

jmp [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT)

;

; (8) Our real working code...

; We come here when they press the next/prev buttons. if we are in a timer repeat

; situation (triggered when they press prev/next for the WEEK/YEAR) then we skip right

; to processing based on the button that was previously pressed

;

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

lda DIGSEL ; Which digit mode are we in?

beq DO\_UPD\_DOW ; 0 - Handle the WEEK

cmp #2

blo DO\_UPD\_MONTH ; <2 = 1 - Handle the MONTH

beq DO\_UPD\_DAY ; 2 - Handle the Day

DO\_UPD\_YEAR ; >2 = 3 - Handle the YEAR

brclr 0,SYSFLAGS,LASTYEAR ; Were we in the down direction?

ldx #99 ; Going up, let the WRAPX routine handle it for us

lda SCAN\_YEAR

jsr [INCA\_WRAPX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#INCA_WRAPX)

bra SAVEYEAR

LASTYEAR

lda SCAN\_YEAR ; Going down, get the year

deca ; Decrement it

bpl SAVEYEAR ; and see if we hit the lower end

lda #99 ; Yes, 2000 wraps down to 1999

SAVEYEAR

sta SCAN\_YEAR ; Save away the new year

bra SETUP\_LAG ; And fire off an event to allow for repeating

DO\_UPD\_DOW ; 0 - Day of week

lda #7 ; We want to iterate 7 times advancing by one day.

sta COUNTER ; (this makes it much easier to handle all the fringe cases)

WEEKLOOP

brclr 0,SYSFLAGS,LASTWEEK ; Are we going backwards?

jsr [INCREMENT\_SCAN\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#INCREMENT_SCAN_DATE) ; Going forwards, advance by one day

bra WEEKLOOPCHK ; And continue the loop

LASTWEEK

jsr [DECREMENT\_SCAN\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#DECREMENT_SCAN_DATE) ; Going backwards, retreat by one day

WEEKLOOPCHK

dec COUNTER ; Count down

tst COUNTER ; See if we hit the limit

bne WEEKLOOP ; and go back for more

; (9) Fake repeater

; This code is used for the Day of week and year modes where we want to have a

; repeating button, but the system routines won’t handle it for us

; It works by posting a USER3 event which has a timer of about ½ second.

; After that timer expires, we get a timer2 event which then repeats every tic.

; The only thing that we have to worry about here is to not go through this

; every time so that it takes ½ second for every repeat.

SETUP\_LAG

brset B\_SCANNING,FLAGBYTE,INLAG ; If we were already scanning, skip out

bset B\_SCANNING,FLAGBYTE ; Indicate that we are scanning

lda #EVT\_USER3 ; and post the event to start it off

jsr [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT)

INLAG

jmp SHOW\_DATE ; Put the date up on the display

; (10) Update routine usage

DO\_UPD\_MONTH ; 1 - Handle the month

lda #MONTH\_JAN ; The bottom end is January

sta UPDATE\_MIN

lda #MONTH\_DEC ; and the top end is December (INCLUSIVE)

sta UPDATE\_MAX

lda #UPD\_HMONTH ; We want the HALF-MONTH udpate function

ldx #SCAN\_MONTH ; To update the SCAN\_MONTH variable

bra SEL\_UPD ; Go do it

DO\_UPD\_DAY ; 2 - Handle the day

lda #1 ; 1 is the first day of the month

sta UPDATE\_MIN

jsr [GET\_SCAN\_MONTHLEN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#GET_SCAN_MONTHLEN) ; Figure out how long the month is

sta UPDATE\_MAX ; and make that the limit

lda #UPD\_HDAY ; We want the HALF-DAY update function

ldx #SCAN\_DAY ; to update the SCAN\_DAY variable

SEL\_UPD

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP) ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

rts

; (11) Making the mode button work

; when they press the mode button, we want to cycle through the various choices

; on the display.

DO\_MODE

lda DIGSEL ; Figure out where we are in the cycle

inca ; advance to the next one

and #3 ; and wrap at 4 to zero

sta DIGSEL

REFRESH

brset B\_CLEAR,FLAGBYTE,NOCLEAR ; Do we need to clear the display first?

FORCEFRESH

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Yes, clear everything before we start

bset B\_CLEAR,FLAGBYTE ; And remember that we have already done that

NOCLEAR

clr BTNFLAGS ; Turn off any scrolling banners

lda #ROW\_TD23 ; Turn off the dash from the week blink

sta DISP\_ROW

bclr COL\_TD23,DISP\_COL

jsr [SHOW\_DATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#SHOW_DATE) ; Display the date

; (12) Establishing a blink routine

; This makes the appropriate section of the display blink based on what we are changing

lda DIGSEL ; Get the digit we are on

beq DO\_BLINK\_DOW ; 0 -> Update Day of week

cmp #2

blo DO\_BLINK\_MONTH ; <2 = 1 -> Update month

beq DO\_BLINK\_DAY ; 2 - Update day of month

DO\_BLINK\_YEAR ; 3: Year

; (13) Calling BLINK\_SECOND

; For BLINK\_SECONDS, the UPDATE\_PARM points to the 2 character format for the year.

ldx SCAN\_YEAR ; Get our year

jsr [GETBCDHI](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#GETBCDHI) ; And extract out the high digit of it

sta YEAR\_DIG1 ; Save that away

ldx SCAN\_YEAR ; Do it again

jsr [GETBCDLOW](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#GETBCDLOW) ; to get the low digit

sta YEAR\_DIG2 ; and save that away

ldx #YEAR\_DIG1 ; the parm points to the first digit

lda #BLINK\_SECONDS ; and we want a BLINK\_SECONDS function

bra SETUP\_BLINK ; so do it already

DO\_BLINK\_DOW ; 0: Day of week:

; (14) Calling BLINK\_SEGMENT

; Unfortunately, there is no blink routine to blink the upper two letters on the display.

; To get around this, I have chosen to blink a single segment on the display (the dash

; after the day of the week). This routine was designed to blink the AM/PM or other

; symbols, but it works quite fine for our purposed. You need to set UPDATE\_POS to have

; the row to be updated and UPDATE\_VAL holds the mask for the COLUMS to be XORed.

; In this way, you might have more than one segment blinking, but there are few segments

; on the same row which would achieve a reasonable effect.

; UPDATE\_POS ROW\_TD23

; UPDATE\_VAL (1<<COL\_TD23)

lda #ROW\_TD23

; We want to blink the DASH after the day of week sta UPDATE\_POS

; Store the ROW for it in UPDATE\_POS lda #(1<<COL\_TD23)

; Get the mask for the column sta UPDATE\_VAL

; And store that in UPDATE\_VAL lda #BLINK\_SEGMENT

; We want a BLINK\_SEGMENT function bra SETUP\_BLINK

; and get to it.

DO\_BLINK\_MONTH ; 1: Month

; (15) Calling BLINK\_HMONTH, BLINK\_HDAY

; These are the normal boring cases of calling the blink routine. They simply need the

; address of the byte holding the value to blink and the function to blink them with.

; UPDATE\_PARM - Points to the month

lda #BLINK\_HMONTH ; We want a BLINK HALF-MONTH function

ldx #SCAN\_MONTH ; to blink our month

bra SETUP\_BLINK ; and do it

DO\_BLINK\_DAY ; 2: Day

; UPDATE\_PARM - Points to the day

lda #BLINK\_HDAY ; We want a BLINK HALF-DAY function

ldx #SCAN\_DAY ; to blink our day

SETUP\_BLINK

jsr [START\_BLINKP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_BLINKP) ; Request the blink function

lda digsel ; Figure out which one we are blinking

lsla ; \*2

lsla ; \*4

lsla ; \*8

add #S8\_WEEK-START ; And use that to index the banner to put on the bottom

jsr [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

bset 2,BTNFLAGS ; Mark a blink routine as pending

rts

;

; (16) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr FLAGBYTE ; start with a clean slate

rts

This is code is built on the [passwd](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\passwd.html) with a quite a few changes and additions.

1. [Program specific constants](#dayfind_1) - different uses for the flags and a couple of new local variables
2. [System entry point vectors](#dayfind_2) - No change here.
3. [Program strings](#dayfind_3) - Gee, we changed the strings.  Note the four strings in a row which serve as help messages when in set mode.
4. [State Table(s)](#dayfind_4) - State table0 is not radically changed (We added the next/prev buttons).  State table 1 is used when we are in the set mode.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.  Note the use of the USER3 event in this table
5. [State Table Handler0](#dayfind_5) - For state0, we only really need to handle the initial enter where we put up the banner.  After a while we time out and put up the current day of the week and our banner.
6. [Get the system date](#dayfind_6) - This shows how to get the current date.
7. [State table 1 handler](#dayfind_7)
8. [Program Specific Code](#dayfind_8) - We use the same UPDATE and BLINK functions from the [Blink](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\blink.html) sample.
9. [Fake Repeater](#dayfind_9) - I’m pretty proud of this one...
10. [Update routine usage](#dayfind_10) - Look here for some clues on using the update routines.
11. [Making the mode button work](#dayfind_11)
12. [Establishing a blink routine](#dayfind_12)
13. [Calling BLINK\_SECOND](#dayfind_13)
14. [Calling BLINK\_SEGMENT](#dayfind_14)
15. [Calling BLINK\_HMONTH, BLINK\_HDAY](#dayfind_15)
16. [Main initialization](#dayfind_16) - Surprisingly, there is not much change here.

## Playing With Sound - TestSnd example

This is a very simple program that I had put together to test out what sounds the watch can make.  The program doesn’t really do a lot except poke the hardware a little.  It does use the update routine without the blinking.  Unlike programs which play a tune, this goes straight to the hardware to test out the capabilities and is completely independent of any sound scheme that you might have loaded.

;Name: Test Sound

;Version: TESTSND

;Description: This routine tests the various sound capabilities of the DataLink.

;by John A. Toebes, VIII

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

CURRENT\_VAL EQU $61

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp DOEVENT0

db TABLE0-TABLE0

L0127: jmp DOEVENT1

db TABLE1-TABLE0

;

; (3) Program strings

S6\_SOUND: timex6 “SOUND “

S6\_TEST: timex6 “ TEST “

S8\_TOEBES: Timex “J.TOEBES”

;

; (4) State Table

;

TABLE0:

db 0

db EVT\_ENTER,TIM\_LONG,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_TIMER2,TIM\_ONCE,0 ;

db EVT\_DNNEXT,TIM\_ONCE,1 ; Next button

db EVT\_DNPREV,TIM\_ONCE,1 ; Prev button

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_DNSET,TIM\_ONCE,0 ; Set button

db EVT\_UPSET,TIM\_ONCE,0 ;

db EVT\_END

TABLE1:

db 1

db EVT\_UPNEXT,TIM\_ONCE,1 ; Releasing the next button

db EVT\_UPPREV,TIM\_ONCE,1 ; Releasing the prev button

db EVT\_USER0,TIM\_ONCE,0 ; Return to the main state table

db EVT\_END ; End of table

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, and RESUME events

;

DOEVENT0:

bset 1,APP\_FLAGS ; Allow us to be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_RESUME ; Did another app get called in the meantime?

beq REFRESH ; We will refresh the display in this case

cmp #EVT\_TIMER2 ; Did the initial timer expire?

beq REFRESH ; Yes, clean up the screen

cmp #EVT\_ENTER ; Is this the initial state?

beq INITBANNER ; Yes, put up the banner

cmp #EVT\_DNSET ; Did they hit the set button

beq PLAYIT

cmp #EVT\_UPSET

beq SILENCE

rts

;

; (6) Sound playing code. Note that we go straight to the hardware here for this one

;

PLAYIT:

lda #ROW\_NOTE ; Turn on the little note symbol

sta DISP\_ROW

bset COL\_NOTE,DISP\_COL

lda CURRENT\_VAL

sta $28

rts

SILENCE:

lda #ROW\_NOTE ; Turn off the little note symbol

sta DISP\_ROW

bclr COL\_NOTE,DISP\_COL

lda #15

sta $28

rts

REFRESH:

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_SOUND-START ; Put “SOUND” on the top of the display

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

ldx CURRENT\_VAL

jsr [FMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34)

bra JBANNER

INITBANNER:

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_SOUND-START ; Put ‘SOUND ‘ on the top line

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_TEST-START ; Put ‘ TEST ‘ on the second line

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

JBANNER

lda #S8\_TOEBES-START

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

bset 7,WRISTAPP\_FLAGS ; Tell them that we are a live application

clr CURRENT\_VAL

rts

;

; (8) State Table 1 Handler

;

; This is called when we press the prev/next button or when the timer fires during that event

;

DOEVENT1:

lda BTNSTATE

cmp #EVT\_DNPREV

beq GO\_DOWN

cmp #EVT\_DNNEXT

beq GO\_UP

lda #EVT\_USER0

jmp [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT)

GO\_DOWN bclr 0,SYSFLAGS ; Mark update direction as down

bra DOUPDN

GO\_UP bset 0,SYSFLAGS ; Mark update direction as up

DOUPDN clra

jsr [CLEARMID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARMID)

sta UPDATE\_MIN

lda #99

sta UPDATE\_MAX

ldx #CURRENT\_VAL

lda #UPD\_MID34

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP)

bset 4,BTNFLAGS

rts

This code has a few notable sections.

1. [Program specific constants](#testsnd_1) - Nothing special here
2. [System entry point vectors](#testsnd_2) - Nothing new here either.
3. [Program strings](#testsnd_3) - Of course we changed the strings once again.
4. [State Table(s)](#testsnd_4) - We have two state tables.  Both of these are pretty simple.  StateTable0 has a lot of values instead of using the EVT\_DNANY event just for a little variety.  StateTable1 is used just for the increment/decrement mode.  See [**The State Table**](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\statetable.html) for a more complete explanation of this.
5. [State Table Handler0](#testsnd_5) - For state0, we only really need to handle the initial enter where we put up the banner.  It times out and puts up the sound banner.  When you press the set button, it will play the sound.
6. [Sound playing code](#testsnd_6) - This code simply pokes the current value to the hardware at $28.  When we let go of the button, we make the hardware silent by poking a $0f to that same location.
7. [Main Initialization routine](#testsnd_7) - Nothing really significant here. This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS).
8. [State Table Handler1](#testsnd_8) - Nothing really significant here, it uses the same update routines that most of the other examples use.

## Using Callbacks - Endoff example

Here is another pretty simple program that shows off a couple of useful features of a wristapp.  This one stems from a request several people have had (including myself) to turn off the alarms on the weekend.  That’s really all this does.  To make it a little more fun, I decided that I wanted to call it “ WEEK “ “ENDOFF”, with the problem that there is no letter K in the character set for the top line on the display.  So, I figured out how to make a reasonably ok looking letter.  You will notice that this program seems to do very little...

;Name: Week End Off

;Version: ENDOFF

;Description: Week End Off - by John A. Toebes, VIII

;This application turns off all alarms on the weekend.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

START EQU \*

;

; (2) System entry point vectors

;

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: jmp CHECKSTATE ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

L0119: jmp ENABLE\_ALL ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

L011c: jmp CHECKSTATE ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

;

S6\_WEEK: timex6 “ WEEH “

S6\_ENDOFF: timex6 “ENDOFF”

S8\_TOEBES: Timex “J.TOEBES”

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM\_LONG,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Allow us to be suspended

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_WEEK-START ; Put ‘ WEEK ‘ on the top line

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_ENDOFF-START ; Put ‘ENDOFF’ on the second line

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

;

; (6) Faking a letter K

;

;

; We have We want it to look like:

; | | |

; | | | |

; | | | |

; |=====| |=====

; | | | |

; | | | |

; | | | |

; This means turning off T5B and turning on T5H

lda #ROW\_T5B

sta DISP\_ROW

bclr COL\_T5B,DISP\_COL

lda #ROW\_T5H

sta DISP\_ROW

bset COL\_T5H,DISP\_COL

jsr CHECKSTATE ; Just for fun, check the alarm state

lda #S8\_TOEBES-START

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

;

; (7) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

bset 7,WRISTAPP\_FLAGS ; Tell them that we are a live application

lda #$C8 ; Bit3 = wristapp wants a call once a day when it changes (WRIST\_DOTIC) (SET=CALL)

; Bit6 = Uses system rules for button beep decisions (SET=SYSTEM RULES)

; Bit7 = Wristapp has been loaded (SET=LOADED)

sta WRISTAPP\_FLAGS

; Fall into CHECKSTATE

;

; (8) Determining the day of the week

;

CHECKSTATE

jsr [ACQUIRE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#ACQUIRE) ; Lock so that it doesn’t change under us

lda TZ1\_DOW ; Assume that we are using the first timezone

jsr [CHECK\_TZ](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CHECK_TZ) ; See which one we are really using

bcc GOT\_TZ1 ; If we were right, just skip on to do the work

lda TZ2\_DOW ; Wrong guess, just load up the second time zone

GOT\_TZ1

jsr [RELEASE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#RELEASE) ; Unlock so the rest of the system is happy

cmp #5 ; Time zone day of week is 0=Monday...6=Sunday

bhs DISABLE\_ALL ; Saturday, Sunday - disable them all

; Fall into ENABLE\_ALL

;---------------------------------------------------------------

; Routine:

; (9) ENABLE\_ALL/DISABLE\_ALL

; Parameters:

; NONE

; Purpose:

; These routines enable/disable all of the alarms. It hides the disabled status of

; the alarm by storing it in bit 3 of the alarm flags.

; Bit0 = Alarm is enabled (SET=ENABLED)

; Bit1 = Alarm is masked (SET=MASKED)

; Bit2 = Current alarm is in 12 hour mode and is in the afternoon (SET=AFTERNOON)

; Bit3 = Alarm was enabled, but we are hiding it (SET=HIDDEN)

; It is safe to call these routine multiple times.

;---------------------------------------------------------------

ENABLE\_ALL

ldx #4 ; We have 5 alarms to go through

ENABLE\_NEXT

lda ALARM\_STATUS,X ; Get the flags for this alarm

lsra ; Shift right 3 to get our hidden bit into place

lsra

lsra

and #1 ; Mask out everything except the hidden bit (now in the enabled position

ora ALARM\_STATUS,X ; Or it back into the flags

and #7 ; and clear out our hidden bit

sta ALARM\_STATUS,X ; then save it out again.

decx ; Count down the number of alarms

bpl ENABLE\_NEXT ; And go back for the next one

rts

DISABLE\_ALL

ldx #4 ; We have 5 alarms to go through

DISABLE\_NEXT

lda ALARM\_STATUS,X ; Get the flags for this alarm

and #1 ; And extract our enabled bit

lsla ; Shift left 3 to save as our hidden bit

lsla

lsla

ora ALARM\_STATUS,X ; Or it back into the flags

and #$0e ; and clear out the enabled bit

sta ALARM\_STATUS,X ; then save it out again.

decx ; Count down the number of alarms

bpl DISABLE\_NEXT ; And go back for the next one

rts

This code has a few notable sections.

1. [Program specific constants](#endoff_1) - We don’t have any
2. [System entry point vectors](#endoff_2) - This is where we have a lot of fun. We are using three of the entry points which we have never used before. The [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC) entry is enabled by us setting bit 3 in the Wristapp\_flags which causes us to get called once a day. While we could enable it to call us hourly, by the minute, or even faster, it really doesn’t make sense to waste processing time. The [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM) entry point gives us a chance to undo our hiding of the alarms just in case the downloaded data wants to mess with it. Lastly, the [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA) entry is called after the data has been loaded into the watch.
3. [Program strings](#endoff_3) - Of course we changed the strings once again. Note that the one string says WEEH and not WEEK since K is not a valid letter in the TIMEX6 alphabet. Don’t worry, we will fix it up at runtime.
4. [State Table(s)](#endoff_4) - We are back to only one state table. In fact, you will see that this state table is even less fancy than the hello world example. We really don’t have any input functions, so we pretty much ignore everything.
5. [State Table Handler0](#endoff_5) - For state0, we only really need to handle the initial enter or resume where we put up the banner.
6. [Faking the letter K](#endoff_6) - All we need to do is turn off one segment and turn on another to turn the H into a K.
7. [Main Initialization routine](#endoff_7) - Nothing really significant here. This is called once when the wristapp is first loaded.  We need to make sure that we set the appropriate bits in [WRISTAPP\_FLAGS](#WRISTAPP_FLAGS). The new bit that we set here is to enable the callback once a day.
8. [Determining the Current Day](#endoff_8) - This really is pretty simple, we figure out the current time zone and grab the day of the week from the right spot.
9. [ENABLE\_ALL/DISABLE\_ALL](#endoff_9) - These routines are pretty simple also, all they have to do is hide the state of the enabled bit in the third bit of the alarm status flags. These routines had to be constructed so that you can call them many times in a row and not lose the original sense of the enabled bit for each alarm. We are able to do that by making sure that we always OR together the bits before clearing out the other.

## Using 3 States - HexDump example

Ok, so you have a computer on your wrist.  What better way to show it off than by having a hex dump utility to traipse through memory.  This is a major overhaul of a previous version of the HexDump application that I have posted.  I have turned it into a real application instead of a simple test program.  It also uses the .ZSM file format to allow you to use it with ASM6805.  You can download it [here](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hexdump.zip)

;Name: Hex Dump

;Version: HEXDUMP

;Description: Hex Dumper - by John A. Toebes, VIII

;This Hex dump routine is a simple thing to test out dumping hex bytes...

;

; Press the NEXT/PREV buttons to advance/backup by 6 bytes of memory at a time

; Press the SET button to change the location in memory where you are dumping.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 0 indicates the direction of the last button

; The other bits are not used

CURRENT\_DIGIT EQU $62

DIGIT0 EQU $63

DIGIT1 EQU $64

DIGIT2 EQU $65

DIGIT3 EQU $66

;

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB0-STATETAB0

L0127: jmp HANDLE\_STATE1

db STATETAB1-STATETAB0

L012b: jmp HANDLE\_STATE2

db STATETAB2-STATETAB0

;

; (3) Program strings

;

S6\_BYTE: timex6 “ BYTE “

S6\_DUMPER: timex6 “DUMPER”

S8\_LOCATION Timex “aaaa “

;

; (4) State Table

;

STATETAB0:

db 0

db EVT\_ENTER,TIM2\_12TIC,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_TIMER2,TIM\_ONCE,0 ; This is the timer

db EVT\_DNNEXT,TIM2\_8TIC,1 ; Next button

db EVT\_DNPREV,TIM2\_8TIC,1 ; Prev button

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_SET,TIM\_ONCE,2 ; Set button

db EVT\_USER0,TIM\_ONCE,$FF ; Return to system

db EVT\_END

STATETAB1:

db 0

db EVT\_UPANY,TIM\_ONCE,0 ; Releasing the prev or next button

db EVT\_TIMER2,TIM2\_TIC,1 ; Repeat operation with a timer

db EVT\_END ; End of table

STATETAB2:

db 2

db EVT\_RESUME,TIM\_ONCE,2 ; Resume from a nested app

db EVT\_DNANY4,TIM\_ONCE,2 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,2 ; NEXT, PREV, SET, MODE button released

db EVT\_USER2,TIM\_ONCE,0 ; Return to state 0

db EVT\_END ; End of table

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_ENTER ; Is this the initial state?

bne SHOWDATA ; no, just clean up the screen

;

; (6) Put up the initial banner screen

;

jsr CLEARALL ; Clear the display

lda #S6\_BYTE-START ; Put ‘ BYTE ‘ on the top line

jsr PUT6TOP

lda #S6\_DUMPER-START ; Put ‘DUMPER’ on the second line

jsr PUT6MID

lda #SYS8\_MODE ; Put MODE on the bottom line

jmp PUTMSGBOT

; (7) FMTHEX is a routine similar to FMTX, but it handles hex values instead

;=======================================================================

; Routine: FMTHEX

; Purpose:

; Format a byte into the buffer

; Parameters:

; A - Byte to be formatted

; X - Offset into Message buffer to put the byte

;=======================================================================

FMTHEX:

sta S8\_LOCATION,X ; Save the byte

and #$0f ; Extract the bottom nibble

sta S8\_LOCATION+1,X ; Save the hex value of the nibble

lda S8\_LOCATION,X ; Get the value once again

lsra ; Shift right by 4 to get the high order nibble

lsra

lsra

lsra

sta S8\_LOCATION,X ; And put it back into the buffer

rts

;

; (8) This is called when we press the prev/next button or when the timer fires during that event

;

HANDLE\_STATE1:

lda BTNSTATE

cmp #EVT\_TIMER2 ; Is this a repeat/timer event?

beq REPEATBTN ; yes, do as they asked

bclr 0,FLAGBYTE ; Assume that they hit the prev button

cmp #EVT\_DNPREV ; Did they hit the prev button

bne REPEATBTN ; Yes, we guessed right

bset 0,FLAGBYTE ; No, they hit next. Mark the direction.

REPEATBTN:

brclr 0,FLAGBYTE,NEXTLOC ; If they hit the next button, go do that operation

;

; They pressed the prev button, let’s go to the previous location

;

PREVLOC:

lda CURRENT\_LOC+1

sub #6

sta CURRENT\_LOC+1

lda CURRENT\_LOC

sbc #0

sta CURRENT\_LOC

bra SHOWDATA

NEXTLOC:

lda #6

add CURRENT\_LOC+1

sta CURRENT\_LOC+1

lda CURRENT\_LOC

adc #0

sta CURRENT\_LOC

;

; (9) This is the main screen update routine.

; It dumps the current memory bytes based on the current address. Note that since it updates the entire

; display, it doesn’t have to clear anything

;

SHOWDATA:

jsr CLEARSYM

clrx

bsr GETBYTE

jsr PUTTOP12

ldx #1

bsr GETBYTE

jsr PUTTOP34

ldx #2

bsr GETBYTE

jsr PUTTOP56

ldx #3

bsr GETBYTE

jsr PUTMID12

ldx #4

bsr GETBYTE

jsr PUTMID34

ldx #5

bsr GETBYTE

jsr PUTMID56

lda CURRENT\_LOC ; Get the high order byte of the address

clrx

bsr FMTHEX ; Put that at the start of the buffer

lda CURRENT\_LOC+1 ; Get the low order byte of the address

ldx #2

bsr FMTHEX ; Put that next in the buffer

lda #S8\_LOCATION-START

jmp BANNER8

; (10) GETBYTE gets a byte from memory and formats it as a hex value

;=======================================================================

; Routine: GETBYTE

; Purpose:

; Read a byte from memory and put it into DATDIGIT1/DATDIGIT2 as hex values

; Parameters:

; X - Offset from location to read byte

; CURRENT\_LOC - Base location to read from

;=======================================================================

GETBYTE

CURRENT\_LOC EQU \*+1 ; Self modifying code... Point to what we want to modify

lda $4000,X ; Get the current byte

sta DATDIGIT2 ; And save it away

lsra ; Extract the high nibble

lsra

lsra

lsra

sta DATDIGIT1 ; And save it

lda DATDIGIT2 ; Get the byte again

and #$0f ; Extract the low nibble

sta DATDIGIT2 ; And save it

rts

;

; (11) State Table 2 Handler

; This is called to process the state events.

; We see SET, RESUME, DNANY4, and UPANY4 events

;

HANDLE\_STATE2:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_UPANY4

beq REFRESH2

cmp #EVT\_DNANY4 ; Is this our initial entry?

bne FORCEFRESH

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_MODE ; Perhaps the MODE button

beq DO\_MODE ; If so, handle it

; It must be the set button, so take us out of this state

bsr SHOWDATA

lda #EVT\_USER2

jmp POSTEVENT

;

; (12) This handles the update routine to change a digit...

;

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

clra

sta UPDATE\_MIN ; Our low end is 0

lda #$F

sta UPDATE\_MAX ; and the high end is 15 (the hes digits 0-F)

bsr GET\_DISP\_PARM

lda #UPD\_DIGIT

jsr START\_UPDATEP ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

rts

;

; (13) This is where we switch which digit we are changing...

;

DO\_MODE

lda CURRENT\_DIGIT

inca

and #3

sta CURRENT\_DIGIT

;

; (14) Refresh the screen and start blinking the current digit...

;

REFRESH2

lda DIGIT0 ; Get the first digit

lsla ; \*16

lsla

lsla

lsla

add DIGIT1 ; Plus the second digit

sta CURRENT\_LOC ; To make the high byte of the address

lda DIGIT2 ; Get the third digit

lsla ; \*16

lsla

lsla

lsla

add DIGIT3 ; Plus the fourth digit

sta CURRENT\_LOC+1 ; To make the low byte of the address

FORCEFRESH

bclr 7,BTNFLAGS ; Turn off any update routine that might be pending

jsr SHOWDATA ; Format the screen

ldx #4 ; We need to copy over 4 bytes from the buffer

COPYIT

decx ; This will be one down.

lda S8\_LOCATION,X ; Get the formatted byte

sta DIGIT0,X ; And store it for the update routine

tstx ; Did we copy enough bytes?

bne COPYIT ; No, go back for more

bsr GET\_DISP\_PARM ; Get the parm for the blink routine

lda #BLINK\_DIGIT ; Request to blink a digit

jsr START\_BLINKP ; And do it

bset 2,BTNFLAGS ; Mark a blink routine as pending

rts

;

; (15) This gets the parameters for an UPDATE/BLINK routine

;

GET\_DISP\_PARM

lda CURRENT\_DIGIT ; Figure out what digit we are dumping

sta UPDATE\_POS ; Store it for the BLINK/UPDATE routine

add #DIGIT0 ; Point to the byte to be updated

tax ; And put it into X as needed for the parameter

rts

;

; (16) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr CURRENT\_DIGIT ; Start out on the first digit

rts

This code has a few notable sections.

1. [Program specific constants](#hexdump_1) - We only really need special storage for the 4 digits which the update/blink routines will handle.
2. [System entry point vectors](#hexdump_2) - We only have a main.  However, we also have 3 state tables.
3. [Program strings](#hexdump_3) - Nothing special here.  We have two strings for the banner and one string that we show the current location with.
4. [State Tables](#hexdump_4) - We have three state tables now.  State table0 does very little other than handle getting into states 1 and 2.  State table 1 is for when you are pressing the prev/next buttons while in the main state to allow you to advance/backup by 6 bytes at a time.  State Table 2 handles all of the setting of the digits.  Note that it would be possible to combine these two states, but it would make the code much more complicated than it needs to be.
5. [State Table 0 Handler](#hexdump_5) - This is actually one of the simplest.  All it has to do is put up the startup banner and then show the current data once that times out.
6. [Initial banner screen](#hexdump_6) - Very simple code to display the name of the application.
7. [FMTHEX](#hexdump_7) is a routine similar to FMTX, but it handles hex values instead.  It is up here in order to allow several of the other BSR instructions to be able to reach the main update routine.  Sometimes moving a subroutine can save you quite a few bytes.
8. [PREV/NEXT Handling](#hexdump_8) This is called when we press the prev/next button or when the timer fires during that event.
9. [Main Update](#hexdump_9) This is the main screen update routine.  Note that we don’t have to refresh anything since the entire screen is written.
10. [GETBYTE](#hexdump_10) gets a byte from memory and formats it as a hex value
11. [State Table 2 Handler](#hexdump_11) - This is very similar to the state handling in the [passwd](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\passwd.html) sample.
12. [Changing Digits](#hexdump_12) This handles the update routine to change a digit...
13. [Switching Digits](#hexdump_13) This is where we switch which digit we are changing...
14. [Blinking Digits](#hexdump_14) Refresh the screen and start blinking the current digit...
15. [GET\_DISP\_PARM](#hexdump_15) This gets the parameters for an UPDATE/BLINK routine.  We made this a subroutine in order to ensure that everything is kept in sync.  It also saves a few bytes.
16. [Main Initialization](#hexdump_16) This is the main initialization routine which is called when we first get the app into memory.  As usual, there is not a lot that we have to do.

## Dumping the EEPROM - promdump example

The HexDump program is great for dumping out the regular memory, but if you search and search, you will never find any of your appointments, lists, phone numbers, or anniversaries in the memory.  That is because they are stored in an EEPROM outside of the address space.  With a few simple modifications to the HexDump program, you can use the system to dump out the contents of the EEPROM.  You can download it [here](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\promdump.zip)

;Name: Prom Dump

;Version: promdump

;Description: Prom Dumper - by John A. Toebes, VIII

;This Prom Dump routine shows you what is in the EEProm

;

; Press the NEXT/PREV buttons to advance/backup by 6 bytes of memory at a time

; Press the SET button to change the location in memory where you are dumping.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

FLAGBYTE EQU $61

; Bit 0 indicates the direction of the last button

; The other bits are not used

CURRENT\_DIGIT EQU $62

DIGIT0 EQU $63

DIGIT1 EQU $64

DIGIT2 EQU $65

DIGIT3 EQU $66

;

; These should have been in the Wristapp.i files, but I forgot them...

;

INST\_ADDRHI EQU $0437

INST\_ADDRLO EQU $0438

HW\_FLAGS EQU $9e

;

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: rts ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

nop

nop

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB0-STATETAB0

L0127: jmp HANDLE\_STATE1

db STATETAB1-STATETAB0

L012b: jmp HANDLE\_STATE2

db STATETAB2-STATETAB0

;

; (3) Program strings

;

S6\_EEPROM: timex6 “EEPROM”

S6\_DUMPER: timex6 “DUMPER”

S8\_LOCATION Timex “aaaa “

;

; (4) State Table

;

STATETAB0:

db 0

db EVT\_ENTER,TIM2\_12TIC,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_TIMER2,TIM\_ONCE,0 ; This is the timer

db EVT\_DNNEXT,TIM2\_8TIC,1 ; Next button

db EVT\_DNPREV,TIM2\_8TIC,1 ; Prev button

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_SET,TIM\_ONCE,2 ; Set button

db EVT\_USER0,TIM\_ONCE,$FF ; Return to system

db EVT\_END

STATETAB1:

db 0

db EVT\_UPANY,TIM\_ONCE,0 ; Releasing the prev or next button

db EVT\_TIMER2,TIM2\_TIC,1 ; Repeat operation with a timer

db EVT\_END ; End of table

STATETAB2:

db 2

db EVT\_RESUME,TIM\_ONCE,2 ; Resume from a nested app

db EVT\_DNANY4,TIM\_ONCE,2 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,2 ; NEXT, PREV, SET, MODE button released

db EVT\_USER2,TIM\_ONCE,0 ; Return to state 0

db EVT\_END ; End of table

CURRENT\_LOC

dw $0000 ; This is where we start in memory

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, TIMER2, and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_ENTER ; Is this the initial state?

bne SHOWDATA ; no, just clean up the screen

;

; (6) Put up the initial banner screen

;

jsr CLEARALL ; Clear the display

lda #S6\_EEPROM-START ; Put ‘EEPROM’ on the top line

jsr PUT6TOP

lda #S6\_DUMPER-START ; Put ‘DUMPER’ on the second line

jsr PUT6MID

lda #SYS8\_MODE ; Put MODE on the bottom line

jmp PUTMSGBOT

; (7) FMTHEX is a routine similar to FMTX, but it handles hex values instead

;=======================================================================

; Routine: FMTHEX

; Purpose:

; Format a byte into the buffer

; Parameters:

; A - Byte to be formatted

; X - Offset into Message buffer to put the byte

;=======================================================================

FMTHEX:

sta S8\_LOCATION,X ; Save the byte

and #$0f ; Extract the bottom nibble

sta S8\_LOCATION+1,X ; Save the hex value of the nibble

lda S8\_LOCATION,X ; Get the value once again

lsra ; Shift right by 4 to get the high order nibble

lsra

lsra

lsra

sta S8\_LOCATION,X ; And put it back into the buffer

rts

;

; (8) This is called when we press the prev/next button or when the timer fires during that event

;

HANDLE\_STATE1:

lda BTNSTATE

cmp #EVT\_TIMER2 ; Is this a repeat/timer event?

beq REPEATBTN ; yes, do as they asked

bclr 0,FLAGBYTE ; Assume that they hit the prev button

cmp #EVT\_DNPREV ; Did they hit the prev button

bne REPEATBTN ; Yes, we guessed right

bset 0,FLAGBYTE ; No, they hit next. Mark the direction.

REPEATBTN:

brclr 0,FLAGBYTE,NEXTLOC ; If they hit the next button, go do that operation

;

; They pressed the prev button, let’s go to the previous location

;

PREVLOC:

lda CURRENT\_LOC+1

sub #6

sta CURRENT\_LOC+1

lda CURRENT\_LOC

sbc #0

sta CURRENT\_LOC

bra SHOWDATA

NEXTLOC:

lda #6

add CURRENT\_LOC+1

sta CURRENT\_LOC+1

lda CURRENT\_LOC

adc #0

sta CURRENT\_LOC

;

; (9) This is the main screen update routine.

; It dumps the current memory bytes based on the current address. Note that since it updates the entire

; display, it doesn’t have to clear anything

;

SHOWDATA:

jsr CLEARSYM

clrx

bsr GETBYTE

jsr PUTTOP12

ldx #1

bsr GETBYTE

jsr PUTTOP34

ldx #2

bsr GETBYTE

jsr PUTTOP56

ldx #3

bsr GETBYTE

jsr PUTMID12

ldx #4

bsr GETBYTE

jsr PUTMID34

ldx #5

bsr GETBYTE

jsr PUTMID56

lda CURRENT\_LOC ; Get the high order byte of the address

clrx

bsr FMTHEX ; Put that at the start of the buffer

lda CURRENT\_LOC+1 ; Get the low order byte of the address

ldx #2

bsr FMTHEX ; Put that next in the buffer

lda #S8\_LOCATION-START

jmp BANNER8

; (10) GETBYTE gets a byte from memory and formats it as a hex value

;=======================================================================

; Routine: GETBYTE

; Purpose:

; Read a byte from memory and put it into DATDIGIT1/DATDIGIT2 as hex values

; Parameters:

; X - Offset from location to read byte

; CURRENT\_LOC - Base location to read from

;=======================================================================

GETBYTE

txa

add CURRENT\_LOC+1

sta INST\_ADDRLO

lda CURRENT\_LOC

adc #0

sta INST\_ADDRHI

bset 6,HW\_FLAGS ; Tell them that it is an EEPROM address

jsr GET\_INST\_BYTE ; Get the current byte

sta DATDIGIT2 ; And save it away

lsra ; Extract the high nibble

lsra

lsra

lsra

sta DATDIGIT1 ; And save it

lda DATDIGIT2 ; Get the byte again

and #$0f ; Extract the low nibble

sta DATDIGIT2 ; And save it

rts

;

; (11) State Table 2 Handler

; This is called to process the state events.

; We see SET, RESUME, DNANY4, and UPANY4 events

;

HANDLE\_STATE2:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_UPANY4

beq REFRESH2

cmp #EVT\_DNANY4 ; Is this our initial entry?

bne FORCEFRESH

lda BTN\_PRESSED ; Let’s see what the button they pressed was

cmp #EVT\_PREV ; How about the PREV button

beq DO\_PREV ; handle it

cmp #EVT\_NEXT ; Maybe the NEXT button?

beq DO\_NEXT ; Deal with it!

cmp #EVT\_MODE ; Perhaps the MODE button

beq DO\_MODE ; If so, handle it

; It must be the set button, so take us out of this state

bsr SHOWDATA

lda #EVT\_USER2

jmp POSTEVENT

;

; (12) This handles the update routine to change a digit...

;

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

bra DO\_UPD

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

clra

sta UPDATE\_MIN ; Our low end is 0

lda #$F

sta UPDATE\_MAX ; and the high end is 15 (the hes digits 0-F)

bsr GET\_DISP\_PARM

lda #UPD\_DIGIT

jsr START\_UPDATEP ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

rts

;

; (13) This is where we switch which digit we are changing...

;

DO\_MODE

lda CURRENT\_DIGIT

inca

and #3

sta CURRENT\_DIGIT

;

; (14) Refresh the screen and start blinking the current digit...

;

REFRESH2

lda DIGIT0 ; Get the first digit

lsla ; \*16

lsla

lsla

lsla

add DIGIT1 ; Plus the second digit

sta CURRENT\_LOC ; To make the high byte of the address

lda DIGIT2 ; Get the third digit

lsla ; \*16

lsla

lsla

lsla

add DIGIT3 ; Plus the fourth digit

sta CURRENT\_LOC+1 ; To make the low byte of the address

FORCEFRESH

bclr 7,BTNFLAGS ; Turn off any update routine that might be pending

jsr SHOWDATA ; Format the screen

ldx #4 ; We need to copy over 4 bytes from the buffer

COPYIT

decx ; This will be one down.

lda S8\_LOCATION,X ; Get the formatted byte

sta DIGIT0,X ; And store it for the update routine

tstx ; Did we copy enough bytes?

bne COPYIT ; No, go back for more

bsr GET\_DISP\_PARM ; Get the parm for the blink routine

lda #BLINK\_DIGIT ; Request to blink a digit

jsr START\_BLINKP ; And do it

bset 2,BTNFLAGS ; Mark a blink routine as pending

rts

;

; (15) This gets the parameters for an UPDATE/BLINK routine

;

GET\_DISP\_PARM

lda CURRENT\_DIGIT ; Figure out what digit we are dumping

sta UPDATE\_POS ; Store it for the BLINK/UPDATE routine

add #DIGIT0 ; Point to the byte to be updated

tax ; And put it into X as needed for the parameter

rts

;

; (16) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

clr CURRENT\_DIGIT ; Start out on the first digit

rts

This code is virtually identical to the promdump example with a few minor changes

1. [Program specific constants](#promdump_1) - I didn’t include these three important addresses in the Wristapp.i file, so you have to define them here.
2. [System entry point vectors](#promdump_2) - No change.
3. [Program strings](#promdump_3) - Of course we change the name of the application.
4. [State Tables](#promdump_4) - No change here.
5. [State Table 0 Handler](#promdump_5) - No change here.
6. [Initial banner screen](#promdump_6) - No change here.
7. [FMTHEX](#promdump_7) - No change here.
8. [PREV/NEXT Handling](#promdump_8) - No change here.
9. [Main Update](#promdump_9) - No change here.
10. [GETBYTE](#promdump_10) This is the only real change. We have to call a system routine to read the byte from memory. Before we do that, we need to store the address into the INST\_ADDR:HI\_INST\_ADDRLO variables and set the HW\_FLAGS bit to indicate that it is an EEPROM address instead of a real memory address. Note that if we clear the bit instead of setting it, this program will behave like the [HEXDUMP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\hexdump.html) program.
11. [State Table 2 Handler](#promdump_11) - No change here.
12. [Changing Digits](#promdump_12) - No change here.
13. [Switching Digits](#promdump_13) - No change here.
14. [Blinking Digits](#promdump_14) - No change here.
15. [GET\_DISP\_PARM](#promdump_15) - No change here.
16. [Main Initialization](#promdump_16) - No change here.

## Tracking Money - Spend Watch example

David Andrews [david@polarnet.com] gets the credit for the inspiration on this example.  Of course it turned out to be a bit harder than I expected to write it - mostly due to the fact that I wanted it to be a full blown wristapp with lots of features yet still fit on the watch.  This one also takes advantage of the ‘parent’ app which allows setting information in the applet without recompiling it.

What was the hardest about this application is making the user interface work and still be intuitive.  Once I got past that, coding was just an exercise left to the reader.

There are a lot of tricks in this code to make it fit.  I created a lot of subroutines and learned some interesting tricks to reduce code size.  It currently sits at 713 bytes and I know how I can get 2 more bytes out of it, but I can’t find much more fluff in the code to cut out.  If you can find ways to make it smaller, I would be more than happy to hear about them...

You can download the wristapp and set program [here](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\spend0.zip)

;Name: spend watch

;Version: spend0

;Description: spend watch - by John A. Toebes, VIII

;This keeps track of how much is in one of 7 categories

;

; Press the NEXT/PREV buttons to advance/backup through the categories

; Press the SET button to add/subtract/set/clear the amounts in the categories

; If you press the set button while the action is blinking, it will be carried out, otherwise

; you can cancel the operation.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

;Parent: SpendSet

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Copyright © 1997 John A. Toebes, VIII \*

;\* All Rights Reserved \*

;\* This program may not be distributed in any form without the permission of the author \*

;\* [jtoebes@geocities.com](mailto:jtoebes@geocities.com) \*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; History:

; 31 July 96 - Corrected problem with totals not being recalculated when you reenter

; the wristapp.

;

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

; We use a few extra bytes here in low memory. Since we can’t possibly

; be running while the COMM app is running, we have no chance of

; conflicting with it’s use of this memory.

;

BLINK\_BUF EQU $5C ; 3 Byte Buffer for the blink routine

; EQU $5D

; EQU $5E

CAT\_SAVE EQU $5F ; Temporary counter variable

COUNTER EQU $60 ; Temporary variable to hold the

FLAGBYTE EQU $61

; Bit 0 indicates that the display does not need to be cleared

; The other bits are not used

CURRENT\_MODE EQU $62 ; The current mode that we are in

MODE\_SELECT EQU 0 ; Set mode, selecting which category to modify

MODE\_HUNDREDS EQU 1 ; Set mode, changing the hundreds of dollars digits

MODE\_DOLLARS EQU 2 ; Set mode, changing the dollars digits

MODE\_CENTS EQU 3 ; Set mode, changing the cents

MODE\_ACTION EQU 4 ; Set mode, changing the action

MODE\_VIEW EQU 5 ; Normal display mode

CATEGORY EQU $63 ; Current category

;

; These three bytes need to be contiguous. The represent the current

; value that is being operated on

;

HUNDREDS EQU $64

DOLLARS EQU $65

CENTS EQU $66

ACTION EQU $67 ; Selector for the current action

ACT\_ADD EQU 0

ACT\_SUB EQU 1

ACT\_SET EQU 2

ACT\_CLEAR EQU 3

AMT\_BASE EQU $F0

;

;

; (2) System entry point vectors

;

START EQU \*

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: jmp DO\_UPD ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

L0119: rts ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

nop

nop

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB0,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE

db STATETAB0-STATETAB0

L0127: jmp HANDLE\_STATE

db STATETAB1-STATETAB0

;

; (3) Program strings

;

; These strings represent the 4 possible actions. They need to be early on in the data segment so that

; then can be pointed to by using 8-bit offset addressing. They are exactly 3 bytes long and are

; displayed by using the BLINK\_TZONE routine

;

S3\_MODE:

S3\_ADD Timex “ADD”

S3\_SUB Timex “SUB”

S3\_SET Timex “SET”

S3\_CLR Timex “CLR”

;

; These are the categories that the end user has configured. They are set by using the SPENDSET program

; which searches for the first string “TOTAL “. These strings must be exactly 8 bytes each in order with

; total being the first one.

;

S8\_TOTAL: Timex “TOTAL “

S8\_CAT1: Timex “CAT1 “

S8\_CAT2: Timex “CAT2 “

S8\_CAT3: Timex “CAT3 “

S8\_CAT4: Timex “CAT4 “

S8\_CAT5: Timex “CAT5 “

S8\_CAT6: Timex “CAT6 “

S8\_CAT7: Timex “CAT7 “

;

; These are the running amounts for each category. Note that you can actually

; initialize them with some default and the code will run properly

;

AMT\_TOTAL: db 0,0,0

AMT\_CAT1: db 0,0,0

AMT\_CAT2: db 0,0,0

AMT\_CAT3: db 0,0,0

AMT\_CAT4: db 0,0,0

AMT\_CAT5: db 0,0,0

AMT\_CAT6: db 0,0,0

AMT\_CAT7: db 0,0,0

;

; These strings prompt for the current mode that we are in. They are displayed on the top line of

; the display.

;

S6\_SELECT timex6 “SELECT”

S6\_AMOUNT timex6 “AMOUNT”

S6\_ACTION timex6 “ACTION”

S6\_SPEND: timex6 “SPEND” ; save a byte by leaching off the space on the start of the next string

S6\_WATCH: timex6 “ WATCH”

;

; This table selects which string is to be displayed. It is directly indexed by the current mode

;

MSG\_TAB db S6\_SELECT-START ; 0 - MODE\_SELECT

db S6\_AMOUNT-START ; 1 - MODE\_HUNDREDS

db S6\_AMOUNT-START ; 2 - MODE\_DOLLARS

db S6\_AMOUNT-START ; 3 - MODE\_CENTS

db S6\_ACTION-START ; 4 - MODE\_ACTION

db S6\_SPEND-START ; 5 - MODE\_VIEW

;

; This is one of the magic tricks for providing the source for the blink routine.

; These are base pointers (offset from HUNDREDS) that we use to copy three bytes into

; BLINK\_BUF. The interesting one here is the MODE\_CENTS entry which points to DATDIGIT1

; This works because the last number that we format happens to be the cents amount,

; and the blink routine expects the two characters instead of the actual value.

;

DATASRC db HUNDREDS-HUNDREDS ; 1 - MODE\_HUNDREDS

db DOLLARS-HUNDREDS ; 2 - MODE\_DOLLARS

db DATDIGIT1-HUNDREDS ; 3 - MODE\_CENTS

db S3\_ADD-HUNDREDS ; 4 - MODE\_ACTION 0 - ACT\_ADD

db S3\_SUB-HUNDREDS ; 4 - MODE\_ACTION 1 - ACT\_SUB

db S3\_SET-HUNDREDS ; 4 - MODE\_ACTION 2 - ACT\_SET

db S3\_CLR-HUNDREDS ; 4 - MODE\_ACTION 3 - ACT\_CLR

;

; This is the parameter to select which blink routine we want to use

;

BLINK\_PARM db BLINK\_MID12 ; 1 - MODE\_HUNDREDS

db BLINK\_MID34 ; 2 - MODE\_DOLLARS

db BLINK\_SECONDS ; 3 - MODE\_CENTS

db BLINK\_TZONE ; 4 - MODE\_ACTION

;

; (4) State Tables

;

; This set of state tables is a little special since we actually use the

; same state processing routine for both states. This saves us a lot of

; memory but still allows us to let the state table make it easy to exit

; the app with the MODE button

;

STATETAB0:

db 0

db EVT\_ENTER,TIM2\_12TIC,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_TIMER2,TIM\_ONCE,0 ; This is the timer

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_SET,TIM\_ONCE,1 ; Set button

db EVT\_DNANY4,TIM\_ONCE,0 ; NEXT, PREV, SET, MODE button pressed

db EVT\_END

STATETAB1:

db 1

db EVT\_RESUME,TIM\_ONCE,1 ; Resume from a nested app

db EVT\_DNANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button pressed

db EVT\_UPANY4,TIM\_ONCE,1 ; NEXT, PREV, SET, MODE button released

db EVT\_USER2,TIM\_ONCE,0 ; Return to state 0

db EVT\_END ; End of table

;

; (5) Put up the initial banner screen

;

HANDLE\_ENTER

clra

sta CATEGORY ; We start out displaying the totals

jsr [FETCH\_CATEGORY](#FETCH_CATEGORY)

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_SPEND-START ; Put ‘SPEND ‘ on the top line

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_WATCH-START ; Put ‘ WATCH’ on the second line

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

clr FLAGBYTE ; Force us to clear the display

lda #MODE\_VIEW ; Start out in the VIEW mode

sta CURRENT\_MODE

lda #SYS8\_MODE ; Put MODE on the bottom line

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

;

; (6) This is the main screen update routine.

;---------------------------------------------------------------

; Routine:

; SHOWCURRENT

; Parameters:

; HUNDREDS,DOLLARS,CENTS - Current value to be displayed

; 0,FLAGBYTE - Screen state (CLR=Must clear it first)

; CATEGORY - the current category to be displayed

; Returns:

; DATDIGIT1,DATDIGIT2 - 2 digit characters for the cents value

; Purpose:

; This routine shows the current selected category and value for the category

;---------------------------------------------------------------

SHOWCURRENT

brset 0,FLAGBYTE,NOCLEAR ; If we don’t need to clear the display, skip it

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

bset 0,FLAGBYTE ; And remember that we did it

NOCLEAR

lda #ROW\_MP45 ; Turn on the decimal point

sta DISP\_ROW

bset COL\_MP45,DISP\_COL

ldx HUNDREDS ; Get the Hundreds

jsr [FMTBLANK0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTBLANK0) ; Format it

jsr [PUTMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID12) ; and display it

;

; We want to output the dollars, but if there were no hundreds, we want to let the

; first digit be a blank. To do this, we simply let it be a blank and set it to a zero

; if there was actually anything in the hundreds field

;

ldx DOLLARS ; Get the Dollars

jsr [FMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX) ; Format it

tst HUNDREDS ; Do we need to have a leading zero?

beq NOBLANKIT ; No, so it is fine

ldx DOLLARS ; Yes, Get the Dollars again

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0) ; And format it with a leading zero

NOBLANKIT

jsr [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34) ; Display the Dollars

ldx CENTS ; Get the Cents

jsr [FMTXLEAD0](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTXLEAD0) ; Format it (and leave it around for later)

jsr [PUTMID56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID56) ; and display it.

lda CATEGORY ; Get which category we want

lsla ; \*2

lsla ; \*4

lsla ; \*8

add #S8\_TOTAL-START ; \*8+the start of the string

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8) ; and display the right string

;

; (7) State Table 0 and 1 Handler

; This is called to process the state events.

; We see SET, RESUME, DNANY4, and UPANY4 events

;

HANDLE\_STATE:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

lda BTNSTATE ; Get the event

cmp #EVT\_ENTER ; Is this the initial state?

beq HANDLE\_ENTER

cmp #EVT\_DNANY4 ; How about a button pressed?

beq HANDLE\_DNANY

bclr 1,BTNFLAGS ; Turn off the repeat counter

cmp #EVT\_SET ; Did they press the set button

bne SKIP2 ; No

clr CURRENT\_MODE ; Yes, Go to MODE\_SELECT

SKIP2 bra GOREFRESH

;

; (8) They pressed a button, so handle it

;

HANDLE\_DNANY

lda BTN\_PRESSED ; Let’s see what the button they pressed was

beq DO\_NEXT ; MODE=1, and NEXT=0, so if it is less, it must be the next button

cmp #EVT\_SET ; MODE=1 SET=2 PREV=3, test all at once

blo DO\_MODE ; <2 = 1 so we have a EVT\_MODE

bhi DO\_PREV ; >2 = 3 so we have a EVT\_PREV

;

; They pressed the set button, so we want to carry out the operation IF they have

; one currently selected.

;

DO\_SETOUT

lda CURRENT\_MODE ; See what mode we were in

cmp #MODE\_ACTION ; Is it the ACTION mode?

bne NO\_ACTION ; No, so just cancel the operation

jsr [DO\_OPERATION](#DO_OPERATION) ; Do what they requested

jsr [DO\_TOTAL](#DO_TOTAL) ; And total up everything

jsr [PLAYCONF](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PLAYCONF) ; Plus tell them that we did it

NO\_ACTION

bclr 0,FLAGBYTE ; We need to clear the display

lda #MODE\_VIEW ; And switch back to VIEW mode

sta CURRENT\_MODE

lda #EVT\_USER2 ; And go back to state 0

jmp [POSTEVENT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#POSTEVENT)

;

; (9) This handles the update routine to change a digit...

;

DO\_NEXT

bset 0,SYSFLAGS ; Mark our update direction as up

BRSKIP2 ; and skip over the next instruction

DO\_PREV

bclr 0,SYSFLAGS ; Mark our update direction as down

DO\_UPD

lda CURRENT\_MODE ; Which mode are we in?

beq CHANGE\_CATEGORY ; 0=MODE\_SELECT, so change the category

cmp #MODE\_VIEW ; 5=MODE\_VIEW, so we also change the category

bne TRYOTHERS

CHANGE\_CATEGORY

; (10) updating the category

ldx #CATEGORY ; Point to the category variable

lda #7 ; get our range of values

bsr [ADJUST\_PX\_ANDA](#ADJUST_PX_ANDA) ; And let the routine do the adjust for us

jsr [FETCH\_CATEGORY](#FETCH_CATEGORY) ; Update the current amount from the new category

GOREFRESH

bra REFRESH

;

; (11) ADJUST\_PX\_ANDA - a routine to adjust a value based on the direction

;---------------------------------------------------------------

; Routine:

; ADJUST\_PX\_ANDA

; Parameters:

; A - Binary range to limit value within ((2\*\*x)-1)

; 0,SYSFLAGS - Direction to adjust, SET=UP

; X - Pointer to value to be adjusted

; Returns:

; Value pointed to by X is adjusted

; Purpose:

; This routine adjusts a value up or down based on the current direction, wrapping

; it to the binary range indicated by the value in A. Note that this value must

; be a power of 2-1 (e.g. 1, 3, 7, 15, 31, 63, or 127)

;---------------------------------------------------------------

ADJUST\_PX\_ANDA

inc ,X

brset 0,SYSFLAGS,NODEC

dec ,X

dec ,X

NODEC and ,X

sta ,X

rts

;

; (12) Try updating one of the other modes

; We have already handled MODE\_SELECT and MODE\_VIEW. This code handles

; MODE\_HUNDREDS, MODE\_DOLLARS, MODE\_CENTS, and MODE\_ACTION

;

TRYOTHERS

cmp #MODE\_CENTS ; 3=MODE\_CENTS

bls TRYMORE ; If it is <=, then we leave only MODE\_ACTION

; (13) updating the Action

lda CATEGORY ; Which category is it?

beq REFRESH ; If we are displaying the total, you can’t change the action

ldx #ACTION ; Point to the current action

lda #3 ; and the range of actions

bsr [ADJUST\_PX\_ANDA](#ADJUST_PX_ANDA) ; and let our simple routine handle it for us

bra REFRESH

TRYMORE

beq DOCENTS ; If it is MODE\_CENTS, go handle it

; (14) Update MODE\_HUNDREDS=1 and MODE\_DOLLARS=2

clrx ; Set the lower limit =0

stx UPDATE\_MIN

ldx #99 ; And the upper limit= 99

stx UPDATE\_MAX

add #HUNDREDS-1 ; Point to the right byte to update

tax ; And put it in X as the parameter

lda CURRENT\_MODE ; MODE=1 MODE=2

deca ; 0 1

lsla ; 0 2

add #UPD\_MID12 ; 5=UPD\_MID12 7=UPD\_MID34

jsr [START\_UPDATEP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_UPDATEP) ; And prepare the update routine

bset 4,BTNFLAGS ; Mark that the update is now pending

rts

;

; (15) This is where we switch which digit we are changing...

;

DO\_MODE

lda CURRENT\_MODE ; Get the mode

ldx #MODE\_ACTION ; Limit it to the first 5 modes

jsr [INCA\_WRAPX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#INCA_WRAPX) ; And let the system increment it for us

sta CURRENT\_MODE ; Save it back

; When we switch to the ACTION mode and we have the Totals category showing,

; we need to limit them to the single action of CLEAR

;

cmp #MODE\_ACTION ; Did we go to action mode?

bne REFRESH ; No, nothing to do

clr ACTION ; Reset the action to be add

tst CATEGORY ; Are we displaying the totals

bne REFRESH ; No, nothing more to do

lda #ACT\_CLEAR ; Yes, switch them to CLEAR

sta ACTION

;

; (16) Refresh the screen and start blinking the current digit...

;

REFRESH

; 0 - SELECT <Category>

; 1 - AMOUNT (Blink hundreds)

; 2 - AMOUNT (Blink dollars)

; 3 - AMOUNT (Blink cents)

; 4 - ACTION

jsr [SHOWCURRENT](#SHOWCURRENT) ; Format the screen

ldx CURRENT\_MODE ; Get the mode

lda MSG\_TAB,X ; So that we can get the message for it

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP) ; And put that on the top of the display

;

; Now we need to make the right thing blink

;

ldx CURRENT\_MODE ; Are we in Select mode?

beq NOBLINK2 ; Yes, don’t blink anything

cpx #MODE\_ACTION ; How about ACTION MODE?

bhi NOBLINK2 ; >ACTION is VIEW mode, so if so, don’t blink either

; 1 -> BLINK\_MID12 PARM=&HUNDREDS

; 2 -> BLINK\_MID34 PARM=&DOLLARS

; 3 -> BLINK\_SECONDS PARM=&2Characters

; 4 -> BLINK\_TZONE PARM=&3Characters

brset 1,BTNFLAGS,NOBLINK2 ; Also, we don’t want to be blinking if we are in an update routine

bne SETUP\_BLINK ; If we were not in action mode, we have the right data source

; Put a > on the display

ldx #C\_RIGHTARR ; Put a > sign right in front of the action

lda #POSL3\_5

jsr [PUTLINE3](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTLINE3)

lda CURRENT\_MODE ; Get the mode

add ACTION ; And add in the action

tax ; To compute our data source pointer

SETUP\_BLINK

;

; (17) Set up the parameters for and call the blink routine

;

ldx DATASRC-1,X ; Get the offsetted pointer to the right data

lda HUNDREDS,X ; And copy the 3 bytes to our blink buffer

sta BLINK\_BUF

lda HUNDREDS+1,X

sta BLINK\_BUF+1

lda HUNDREDS+2,X

sta BLINK\_BUF+2

ldx CURRENT\_MODE ; Get our mode again

lda BLINK\_PARM-1,X ; and use it to pick up which parameter we are passing

ldx #BLINK\_BUF ; Point to the common blink buffer

jsr [START\_BLINKP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#START_BLINKP) ; And do it

bset 2,BTNFLAGS ; Mark a blink routine as pending

NOBLINK2

rts

;

; (18) Update MODE\_CENTS

; This is a special case since we don’t have a system routine that allows updating

; the right most digits on the middle line. Fortunately we can fake it by turning

; on the tic timer and waiting until 8 tics have passed before going into a repeat

; loop. The code has been carefully constructed so that the tic timer can just go

; straight to the DO\_UPD code to work.

DOCENTS

ldx #COUNTER ; Point to the counter (saves code size)

brset 1,BTNFLAGS,NOSTART ; Are we already in an update loop?

lda #8 ; No, we need to wait 8 tics

sta ,X ; X->COUNTER ; Save the value

BSET 1,BTNFLAGS ; and start the timer

bra DOIT ; But still do it once right now

;

DEC\_DELAY

dec ,X ; X->COUNTER ; We haven’t hit the limit, decrement it and try again

rts

NOSTART

tst ,X ; X->COUNTER ; We are in the loop, have we hit the limit?

bne DEC\_DELAY ; no, go off and delay once more

DOIT

lda #99 ; Our upper limit is 99

ldx #CENTS ; Point to the cents variable (saves code size)

brset 0,SYSFLAGS,UPCENTS ; Are we in an up mode?

dec ,X ; X->CENTS ; Down, decrement the value

bpl REFRESH ; If we didn’t wrap, just go display it

sta ,X ; X->CENTS ; We wrapped, save the upper limit

bra REFRESH ; and go display it

UPCENTS

inc ,X ; X->CENTS ; Up, increment the value

cmp ,X ; X->CENTS ; Did we hit the limit?

bpl REFRESH ; No, go display it

clr ,X ; X->CENTS ; Yes, wrap to the bottom

bra REFRESH ; and display it

;

; (19) DO\_OPERATION - Perform the requested operation

;---------------------------------------------------------------

; Routine:

; DO\_OPERATION

; Parameters:

; HUNDREDS,DOLLARS,CENTS - Amount to be added/subtracted/set

; CATEGORY - Item to be updated

; ACTION - 0 = ACT\_ADD

; 1 = ACT\_SUB

; 2 = ACT\_SET

; 3 = ACT\_CLEAR

; Purpose:

; Adjusts the corresponding category by the given amount

;---------------------------------------------------------------

DO\_OPERATION

lda CATEGORY ; Get our category

bsr [COMPUTE\_BASE](#COMPUTE_BASE) ; And point to the data for it

lda ACTION ; Which action is it?

beq DO\_ADD ; 0=ADD, go do it

cmp #ACT\_SET ; 3 way compare here... (code trick)

beq DO\_SET ; 2=SET, go do it

blo DO\_SUB ; <2=1 (SUB), go do it

DO\_CLR ; >2 = 3 (CLEAR)

clr HUNDREDS ; Clear out the current values

clr DOLLARS

clr CENTS

tst CATEGORY ; Were we clearing the total?

bne DO\_SET ; No, just handle it

;

; They want to clear everything

;

ldx #(3\*8)-1 ; Total number of categories

CLEAR\_TOTALS

; Mini Routine here X=number of bytes to clear

clra

CLR\_MORE

sta AMT\_TOTAL,X ; Clear out the next byte

decx ; Decrement the number to do

bpl CLR\_MORE ; And go for more

rts

;

; (20) Handle Subtracting a value

;

DO\_SUB

neg HUNDREDS ; Just negate the value to be added

neg DOLLARS

neg CENTS ; And fall into the add code

;

; (21) Handle Adding a value

;

DO\_ADD

lda CENTS ; Add the cents

add AMT\_BASE+2,X

sta CENTS

lda DOLLARS ; Add the dollars

add AMT\_BASE+1,X

sta DOLLARS

lda HUNDREDS ; Add the hundreds

add AMT\_BASE,X

sta HUNDREDS

ldx #CENTS ; Point to the cents as it will be the first one we fix up

tst ACTION ; See what type of operation we just did

beq FIXUP\_ADD ; Was it an ADD? If so, do do it

bsr TRYDEC ; Decrement, fix up the Cents

bsr TRYDEC ; Then fix up the dollars

lda HUNDREDS ; Did the hundreds underflow as a result?

bmi DO\_CLR ; Yes, so just set everything to zero

bra DO\_SET ; No, so copy over the values to the current entry

TRYDEC

lda ,X ; Get the current byte to check

bpl RETDEC ; If it didn’t underflow, then skip to the next byte

add #100 ; Add back the 100 that it underflowed

sta ,X ; And save that away

decx ; Back up to the next most significant byte

dec ,X ; and borrow the one

rts

RETDEC decx ; No need to do anything, so skip to the next byte

rts

TRYADD

lda ,X ; Get the current byte to check

sub #100 ; See if it was less than 100

bmi RETDEC ; If so, then it was already normalized so skip out

sta ,X ; It was an overflow, so save the fixed value

decx ; Skip to the next byte

inc ,X ; And add in the overflow

rts

FIXUP\_ADD

bsr TRYADD ; Fix up the cents

bsr TRYADD ; and then fix up the dollars

;

; (22) Handle setting a value

;

DO\_SET

bsr [COMPUTE\_CATEGORY\_BASE](#COMPUTE_CATEGORY_BASE) ; Point to the data for our category

lda HUNDREDS ; Copy over the values to the current category

sta AMT\_BASE,X

lda DOLLARS

sta AMT\_BASE+1,X

lda CENTS

sta AMT\_BASE+2,X

rts

;

; (23) COMPUTE\_BASE - Computes an offset pointer to get to the total amounts

; This is a trick to save us a few bytes in the instructions.

;---------------------------------------------------------------

; Routine:

; COMPUTE\_BASE

; Parameters:

; A - Offset into total

; Returns:

; X - Pointer relative to AMT\_BASE to use

; Purpose:

; Computes an offset pointer to get to the total amounts

;---------------------------------------------------------------

COMPUTE\_CATEGORY\_BASE

lda CATEGORY ; Get our category

COMPUTE\_BASE

ldx #3

mul

add #AMT\_TOTAL-AMT\_BASE

tax

rts

;

; (24) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

; Fall into DO\_TOTAL

;

; (25) DO\_TOTAL - Recomputes the current total

;---------------------------------------------------------------

; Routine:

; DO\_TOTAL

; Parameters:

; NONE

; Purpose:

; Recomputes the current total

;---------------------------------------------------------------

DO\_TOTAL

lda CATEGORY ; Remember our category

sta CAT\_SAVE

clr ACTION ; Say that we want to add 0=ACT\_ADD

clr CATEGORY ; To the total category

ldx #2 ; But we need to clear it first

bsr [CLEAR\_TOTALS](#CLEAR_TOTALS)

lda #7 ; And iterate over the 7 categories

sta COUNTER

TOT\_LOOP

lda COUNTER ; Get our current category

bsr [FETCH\_CATEGORY](#FETCH_CATEGORY) ; And fetch the data

jsr [DO\_OPERATION](#DO_OPERATION) ; Then add it to the total

dec COUNTER ; Go to the next category

bne TOT\_LOOP ; Until we are done

lda CAT\_SAVE ; Restore the category

sta CATEGORY

; fall into FETCH\_CATEGORY

; (26) FETCH\_CATEGORY - Retrieves the value of the total amount for the selected category

;---------------------------------------------------------------

; Routine:

; FETCH\_CATEGORY

; Parameters:

; A - Category to be fetched

; Returns:

; HUNDREDS,DOLLARS,CENTS - Current value of selected category

; Purpose:

; Retrieves the value of the total amount for the selected category

;---------------------------------------------------------------

FETCH\_CATEGORY

bsr [COMPUTE\_BASE](#COMPUTE_BASE) ; Get the pointer to the base

lda AMT\_BASE,X ; And retrieve the data

sta HUNDREDS

lda AMT\_BASE+1,X

sta DOLLARS

lda AMT\_BASE+2,X

sta CENTS

rts

;--------------------END OF CODE---------------------------------------------------

This is a pretty significant program and the sections are ordered to make the branches all work out. Here’s a quick look around at the sections.

1. [Program specific constants](#spend_1) - It is worth noting that in this case, I actually intruded on the space which one might consider reserved for the system applications. However, the only one that uses any of this memory is the Comm app and there is no chance that we need to be running while it is. We are forced in several instances to use this lower memory because the system roms need a pointer passed in X. Since our code loads into 0110 and beyond, we have to use lower memory if we want to actually point to something.
2. [System entry point vectors](#spend_2) - Nothing really special here. However, we do have a timer routine that we enable when we are inputting cents. What is nice in this case is that the code is constructed so that it jumps right into the processing loop to act as if a timer event had occurred with the normal state processing.
3. [Program strings](#spend_3) - We have quite a few strings that we have created. We also take advantage of table of pointers to save us code space.
4. [State Tables](#spend_4) - This is a pretty unusual program in that even though we have two state tables, they both point to the same state table processing routine. This allows me to let the system handle knowing when we are in set mode to allow for the mode button to advance us through states in the set mode and to take us out of the wristapp when we are not in set mode.
5. [Initial Banner Screen](#spend_5) - No real surprises here.
6. [This is the main screen update routine.](#spend_6)
7. [State Table 0 and 1 Handler](#spend_7)
8. [They pressed a button, so handle it](#spend_8)
9. [This handles the update routine to change a digit...](#spend_9)
10. [updating the category](#spend_10)
11. [ADJUST\_PX\_ANDA - a routine to adjust a value based on the direction](#spend_11)
12. [Try updating one of the other modes](#spend_12)
13. [updating the Action](#spend_13)
14. [Update MODE\_HUNDREDS=1 and MODE\_DOLLARS=2](#spend_14)
15. [This is where we switch which digit we are changing...](#spend_15)
16. [Refresh the screen and start blinking the current digit...](#spend_16)
17. [Set up the parameters for and call the blink routine](#spend_17)
18. [Update MODE\_CENTS](#spend_18)
19. [DO\_OPERATION - Perform the requested operation](#spend_19)
20. [Handle Subtracting a value](#spend_20)
21. [Handle Adding a value](#spend_21)
22. [Handle setting a value](#spend_22)
23. [COMPUTE\_BASE - Computes an offset pointer to get to the total amounts](#spend_23)
24. [This is the main initialization routine which is called when we first get the app into memory](#spend_24)
25. [DO\_TOTAL - Recomputes the current total](#spend_25)
26. [FETCH\_CATEGORY - Retrieves the value of the total amount for the selected category](#spend_26)
27. [Handle the underflows when adding dollars and cents](#spend_27)

## Creating a Sound Scheme - Sound1 example

With a little prodding, I decided to update the assembler so that allows you to create a sound scheme automatically. This is a very simple sound scheme which gives you the same sounds as the Datalink default ones.  Use this as a basis to create any new ones that you might want.

;Sound: Datalink Default

;Version: Sound1

;

; This sample corresponds to the default sounds that you get when you reset a DataLink

; watch to its default state.

;

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Copyright © 1997 John A. Toebes, VIII \*

;\* All Rights Reserved \*

;\* This program may not be distributed in any form without the permission of the author \*

;\* [jtoebes@geocities.com](mailto:jtoebes@geocities.com) \*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

INCLUDE “WRISTAPP.I”

;

; This is the default sound table

;

DEF\_SOUNDS

db SP\_1-SD\_1 ; 0000: 08

db SD\_1-DEF\_SOUNDS ; 0001: 0b BUTTON BEEP

db SD\_2-DEF\_SOUNDS ; 0002: 0c RETURN TO TIME

db SD\_3-DEF\_SOUNDS ; 0003: 0d HOURLY CHIME

db SD\_4-DEF\_SOUNDS ; 0004: 0e CONFIRMATION

db SD\_5-DEF\_SOUNDS ; 0005: 0f APPOINTMENT BEEP

db SD\_5-DEF\_SOUNDS ; 0006: 0f ALARM BEEP

db SD\_5-DEF\_SOUNDS ; 0007: 0f PROGRAM DOWNLOAD

db SD\_5-DEF\_SOUNDS ; 0008: 0f EXTRA

db SD\_6-DEF\_SOUNDS ; 0009: 11 COMM ERROR

db SD\_7-DEF\_SOUNDS ; 000a: 12 COMM DONE

;

; This is the [soundlet count table](#Soundlet_count_table) which contains the duration

; counts for the individual soundlets

;

SD\_1 db SND\_END+1 ; 000b: 81

SD\_2 db SND\_END+1 ; 000c: 81

SD\_3 db SND\_END+2 ; 000d: 82

SD\_4 db SND\_END+4 ; 000e: 84

SD\_5 db 10,SND\_END+40 ; 000f: 0a a8

SD\_6 db SND\_END+10 ; 0011: 8a

SD\_7 db SND\_END+32 ; 0012: a0

;

; This is the [soundlet pointer table](#Soundlet_pointer_table) which contains the pointers to the soundlets

;

SP\_1 db SL\_2-DEF\_SOUNDS ; 0013: 1d

SP\_2 db SL\_1-DEF\_SOUNDS ; 0014: 1b

SP\_3 db SL\_3-DEF\_SOUNDS ; 0015: 1f

SP\_4 db SL\_2-DEF\_SOUNDS ; 0016: 1d

SP\_5 db SL\_4-DEF\_SOUNDS ; 0017: 22

db SL\_5-DEF\_SOUNDS ; 0018: 27

SP\_6 db SL\_6-DEF\_SOUNDS ; 0019: 2a

SP\_7 db SL\_2-DEF\_SOUNDS ; 001a: 1d

;

; These are the [soundlets](#soundlet) themselves. The +1 or other number

; indicates the duration for the sound.

;

SL\_1 db TONE\_HI\_GSHARP+1 ; 001b: 91

db TONE\_END ; 001c: 00

SL\_2 db TONE\_MID\_C+1 ; 001d: 31

db TONE\_END ; 001e: 00

SL\_3 db TONE\_MID\_C+2 ; 001f: 32

db TONE\_PAUSE+2 ; 0020: f2

db TONE\_END ; 0021: 00

SL\_4 db TONE\_HI\_C+2 ; 0022: 22

db TONE\_PAUSE+2 ; 0023: f2

db TONE\_HI\_C+2 ; 0024: 22

db TONE\_PAUSE+10 ; 0025: fa

db TONE\_END ; 0026: 00

SL\_5 db TONE\_HI\_C+2 ; 0027: 22

db TONE\_PAUSE+2 ; 0028: f2

db TONE\_END ; 0029: 00

SL\_6 db TONE\_HI\_C+3 ; 002a: 23

db TONE\_MID\_C+3 ; 002b: 33

db TONE\_END ; 002c: 00

;

; This is the tone that the comm app plays for each record

;

db TONE\_MID\_C/16 ; 002d: 03

## Random Numbers and Marquis - 3Ball example

Wayne Buttles contributed the first version of this Wristapp which gives you a simple decision maker. It inspired me to make a few adjustments to it and add a real random number generator that you can use. I’ve also included a little busy wait Marquis while it is selecting a number to show off a use of the time. This Wristap also illustrates that you don’t always have to put a JMP or RTS instruction in the entry point vectors.

;Name: 3BALL

;Version: 3BALL

;Description: An executive decision maker that will give a yes/no/maybe answer. Pressing Next will generate another answer and beep (since it will be the same answer sometimes).

;

;© 1997 Wayne Buttles ([timex@fdisk.com)](mailto:timex@fdisk.com)). Compiled using tools and knowledge published by John A. Toebes, VIII and Michael Polymenakos ([mpoly@panix.com)](mailto:mpoly@panix.com)).

; Some enhancements by John Toebes...

;

;HelpFile: watchapp.hlp

;HelpTopic: 100

;

; (1) Program specific constants

;

INCLUDE “WRISTAPP.I”

;

; Program specific constants

;

CURRENT\_TIC EQU $27 ; Current system clock tic (Timer)

LAST\_ANS EQU $61

RAND\_SEED EQU $60

MARQ\_POS EQU $62

START EQU \*

;

; (2) System entry point vectors

;

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: bclr 1,BTNFLAGS ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

rts

L0116: jmp FLASH ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

L0119: bclr 1,BTNFLAGS ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

rts

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

;

S6\_MSG timex6 “3 BALL”

S6\_MAYBE timex6 “MAYBE”

S6\_YES timex6 “ YES”

S6\_NO timex6 “ NO”

S6\_MARQ timex6 “ +O+ “

MARQ\_SEL

DB S6\_MARQ+2-START

DB S6\_MARQ+3-START

DB S6\_MARQ+2-START

DB S6\_MARQ+1-START

DB S6\_MARQ-START

DB S6\_MARQ+1-START

MSG\_SEL DB S6\_YES-START

DB S6\_NO-START

DB S6\_MAYBE-START

DB S6\_YES-START

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM2\_16TIC,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_DNNEXT,TIM2\_16TIC,0 ; Next button

db EVT\_TIMER2,TIM\_ONCE,0 ; Timer

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, RESUME, TIMER2 and NEXT events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

bclr 1,BTNFLAGS ; Turn off the MARQUIS tic event

lda BTNSTATE

cmp #EVT\_DNNEXT ; Did they press the next button?

beq DOITAGAIN

cmp #EVT\_ENTER ; Or did we start out

beq DOITAGAIN

cmp #EVT\_RESUME

beq REFRESH

;

; (6) Select a random answer

;

SHOWIT

bsr [RAND](#3ball_9)

and #3 ; go to a 1 in 4 chance

sta LAST\_ANS

;

; (7) Display the currently selected random number

;

REFRESH

ldx LAST\_ANS ; Get the last answer we had, and use it as an index

lda MSG\_SEL,X ; And get the message to display

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP) ; Put that on the top

BANNER

lda #S6\_MSG-START

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

lda #SYS8\_MODE ; And show the mode on the bottom

jmp [PUTMSGBOT](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMSGBOT)

;

; (8) This flashes the text on the screen

;

FLASH

lda CURRENT\_APP ; See which app is currently running

cmp #APP\_WRIST ; Is it us?

bne L0113 ; No, so just turn off the tic timer since we don’t need it

ldx #5

lda MARQ\_POS

jsr INCA\_WRAPX

sta MARQ\_POS

tax

lda MARQ\_SEL,X

jmp PUT6TOP

;

; (9) They want us to do it again

;

DOITAGAIN ; Tell them we are going to do it again

clr MARQ\_POS

bset 1,BTNFLAGS

bra BANNER

;

; (10) Here is a simple random number generator

;

RAND

lda RAND\_SEED

ldx #85

mul

add #25

sta RAND\_SEED

rola

rola

rola

rts

;

; (11) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

lda CURRENT\_TIC

sta RAND\_SEED

rts

1. [Program specific constants](#3ball_1) - We have two variables - RAND\_SEED and CURRENT\_TIC which we use for the random number routine.  RAND\_SEED is used to keep track of the last random number returned so that we continue to deliver random numbers.  CURRENT\_TIC is what is set by the system when it reads the clock to keep the watch time up to date.  We use it once to provide a seed for the random number generator.
2. [System entry point vectors](#3ball_2) - This one gets to be a little fun.  Notice for the [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND) and [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM) routines that we don’t have a JMP instruction, but instead put the actual code in line.  This saves use a couple of bytes.
3. [Program strings](#3ball_3) - We are pretty frugal here in reusing blanks at the end of the string very liberally.  Also note the S6\_MARQ string which has blanks at the start and end so that it can shuffle left and right on the display but always have blanks visible.  The MARQ\_SEL and MSG\_SEL tables are simply offsets that allow us to select the message with a simple load instruction instead of having to calculate the offset.
4. [State Table](#3ball_4) - This is pretty vanilla here except for the fact that we have a very long time interval after the DNNEXT and ENTER events.  It is during this time that the Marquis runs.  We could make it even longer, but this seems to be a good compromise between seeing something happen and actually getting a result in a reasonable time.
5. [State Table 0 Handler](#3ball_5) - Extremely simple, there are only four events that we want to see and this is the typical test and branch one.  The only unique thing here is that we turn off the Marquis timer as soon as we get any event.
6. [Select a random answer](#3ball_6) - As if life weren’t complicated enough.  This is where we go when it is time to make a decision.  For this we get a random number and limit it to 1 in four.
7. [Display the currently selected random number](#3ball_7) - Given a random number, we just get the message for it and put it on the display.
8. [This flashes the text on the screen](#3ball_8) - This is the cheap way to do a Marquis.  Just have a string wider than the display and change the offset from the start at which you start to display.  For this one, there are only 6 states and we select the starting offset from the table based on our current cycle.  Note that this routine is called by the TIC timer which is enabled when they want a new random number.  Eventually the timer for the main event will run out and they will simply stop calling us.
9. [They want us to do it again](#3ball_9) - Whenever we want to do a new random number, we just start the Marquis tic timer and set up the display.
10. [Here is a simple random number generator](#3ball_10) - This is a random number generator that you might want to use.  It is a derivative of the typical calculation rand = (seed\*25173 + 13849) MOD 65536 which I have chopped down to fit in the 8 bit world as rand = (seed \* 85 + 25) MOD 256.  Because the low order bits do produce a pattern cycle which is fairly predictable, we rotate through to get a few of the more randomly occurring bits.
11. [This is the main initialization routine which is called when we first get the app into memory](#3ball_11) - Very boring stuff here, but we do take a moment to initialize the random number seed with the current tic count just to make it a little more variable.

## Playing Hourly Chimes - Ships Bells example

Theron E. White, CPA” <[twhite@mercury.peganet.com](mailto:twhite@mercury.peganet.com)> suggested a wristapp to allow the hourly chimes to play the number of bells past a shift change.  This would be 8 bells at midnight, 8AM, and 4PM, 1 bell at 1AM, 9AM, and 5PM, with one more bell for each hour after that.  This wristapp is a little unique in that it doesn’t use the sound playing routines directly, but instead goes straight to the hardware.  This allows you to have whatever sound scheme you want in the watch.  The pattern for the bells and the actual tone is customizable below.  This app is also a good candidate for combining with another wristapp as this one has no real user input operations.

;Name: Ships Bells

;Version: SHIPBELL

;Description: Ships bells - by John A. Toebes, VIII

;This application turns makes the hour chime with nautical bells.

;

;TIP: Download your watch faster: Download a WristApp once, then do not send it again. It stays in the watch!

;HelpFile: watchapp.hlp

;HelpTopic: 106

INCLUDE “WRISTAPP.I”

;

; (1) Program specific constants

;

START EQU \*

CHANGE\_FLAGS EQU $92 ; System Flags

SND\_POS EQU $61

SND\_REMAIN EQU $62

SND\_NOTE EQU $63

NOTE\_PAUSE EQU (TONE\_PAUSE/16)

NOTE\_BELL EQU (TONE\_MID\_C/16)

;

; (2) System entry point vectors

;

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: rts ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

nop

nop

L0116: jmp CHECKSTATE ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

L0119: jmp STOPIT ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

L011c: rts

nop

nop ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

;

S6\_SHIPS: timex6 “SHIPS”

S6\_BELLS: timex6 “ BELLS”

S8\_TOEBES: Timex “J.TOEBES”

;

; Here is the pattern for the ships bells. We want to have a short bell followed by a very short silence

; followed by a longer bell. We use 3 tics for the short bell, 1 tic for the silence and 6 tics for the longer

; bell. The last bell is 7 ticks.

; We then have to byte swap each of these because the BRSET instruction numbers from bottom to top.

;

; The string looks like:

; 111 0 111111 000000 111 0 111111 000000 111 0 111111 000000 111 0 111111 000000

; Taking this into clumps of 4 bytes, we get

; 1110 1111 1100 0000 1110 1111 1100 0000 1110 1111 1100 0000 1110 1111 1100 0000 1111 1110

;

Pattern DB $F7 ;1110 1111 ; 8 start here

DB $03 ;1100 0000

P67 DB $F7 ;1110 1111 ; 6, 7 start here

DB $03 ;1100 0000

P45 DB $F7 ;1110 1111 ; 4, 5 start here

DB $03 ;1100 0000

P23 DB $F7 ;1110 1111 ; 2, 3 start here

DB $03 ;1100 0000

P1 DB $7F ;1111 1110 ; 1 starts here

;

; This table indexes where we start playing the tone from

;

STARTS

DB (Pattern-Pattern)\*8 ; 0 (8 AM, 4PM, Midnight)

DB (P1-Pattern)\*8 ; 1 (1 AM, 9AM, 5PM)

DB (P23-Pattern)\*8 ; 2 (2 AM, 10AM, 6PM)

DB (P23-Pattern)\*8 ; 3 (3 AM, 11AM, 7PM)

DB (P45-Pattern)\*8 ; 4 (4 AM, NOON, 8PM)

DB (P45-Pattern)\*8 ; 5 (5 AM, 1PM, 9PM)

DB (P67-Pattern)\*8 ; 6 (6 AM, 2PM, 10PM)

DB (P67-Pattern)\*8 ; 7 (7 AM, 3PM, 11PM)

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM\_LONG,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

;

; (5) State Table 0 Handler

; This is called to process the state events.

; We see ENTER and RESUME events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Allow us to be suspended

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL) ; Clear the display

lda #S6\_SHIPS-START ; Put ‘SHIPS ‘ on the top line

jsr [PUT6TOP](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6TOP)

lda #S6\_BELLS-START ; Put ‘ BELLS’ on the second line

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

bsr FORCESTATE ; Just for fun, check the alarm state

lda #S8\_TOEBES-START

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

;

; (6) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$C4 ; Bit2 = wristapp wants a call once an hour when it changes (WRIST\_DOTIC) (SET=CALL)

; Bit6 = Uses system rules for button beep decisions (SET=SYSTEM RULES)

; Bit7 = Wristapp has been loaded (SET=LOADED)

sta WRISTAPP\_FLAGS

bclr 2,MODE\_FLAGS ; Turn off the hourly chimes

clr SND\_REMAIN

;

; (7) Determining the current hour

;

CHECKSTATE

brclr 5,CHANGE\_FLAGS,NO\_HOUR ; Have we hit the hour mark?

FORCESTATE

bclr 3,MAIN\_FLAGS ; Make sure we don’t play the system hourly chimes

jsr [ACQUIRE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#ACQUIRE) ; Lock so that it doesn’t change under us

lda TZ1\_HOUR ; Assume that we are using the first timezone

jsr [CHECK\_TZ](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CHECK_TZ) ; See which one we are really using

bcc GOT\_TZ1 ; If we were right, just skip on to do the work

lda TZ2\_HOUR ; Wrong guess, just load up the second time zone

GOT\_TZ1

;

; 12 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12

; 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18

; deca FF 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17

; anda 07 00 01 02 03 04 05 06 07 00 01 02 03 04 05 06 07 00 01 02 03 04 05 06 07

and #7 ; Convert the hour to the number of bells

tax ; Save away as an index into the start position table

bne NOTEIGHT ; Is it midnight (or a multiple of 8)

lda #8 ; Yes, so that is 8 bells, not zero

NOTEIGHT

lsla ; Multiple the number of bells by 8 to get the length

lsla

lsla

sta SND\_REMAIN ; Save away the number of bells left to play

lda STARTS,X ; Point to the pattern of the first bell

sta SND\_POS

bset 1,BTNFLAGS ; Turn on the tic timer

JMP [RELEASE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#RELEASE) ; And release our lock on the time

;

; (8) Playing the next note piece

;

NO\_HOUR

lda SND\_REMAIN ; Do we have any more notes to play?

bne DO\_SOUND ; No, skip out

STOPIT

lda #TONE\_PAUSE ; End of the line, shut up the sound hardware

sta $28

clr SND\_REMAIN ; Force us to quit looking at sound

bclr 1,BTNFLAGS ; and turn off the tic timer

rts

DO\_SOUND

deca ; Yes, note that we used one up

sta SND\_REMAIN

lda SND\_POS ; See where we are in the sound

lsra ; Divide by 8 to get the byte pointer

lsra

lsra

tax ; and make it an index

lda Pattern,X ; Get the current pattern byte

sta SND\_NOTE ; And save it where we can test it

lda SND\_POS ; Get the pointer to where we are in the sound

inc SND\_POS ; Advance to the next byte

and #7 ; and hack off the high bytes to leave the bit index

lsla ; Convert that to a BRSET instruction

sta TSTNOTE ; And self modify our code so we can play

TSTNOTE brset 0,SND\_NOTE,PLAYIT ; If the note is not set, skip out

lda #TONE\_PAUSE ; Not playing, we want to have silence

brskip2

PLAYIT lda #NOTE\_BELL ; Playing, select the bell tone

sta $28 ; And make it play

NO\_SOUND

rts

1. [Program specific constants](#shipbell_1) - We define the CHANGE\_FLAGS because it is not currently in Wristapp.i.  This allows us to turn off the system attempts at playing hourly chimes.  We also select the tone that we want to play the bells with.  This seems to work as the best one to be heard as bells.
2. [System entry point vectors](#shipbell_2) - The only interesting thing here is that we use the [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM) entry to disable any bell playing that might have started.
3. [Program strings](#shipbell_3) - The pattern and starts tables are used to describe when we will be playing notes and when we will be pausing.
4. [State Table](#shipbell_4) - Pretty boring here.
5. [State Table 0 Handler](#shipbell_5) - Also amazingly boring.  The only interesting thing that we do here is to force the current bells to play when you enter the app.
6. [Main initialization routine](#shipbell_6) - Nothing spectacular here, other than the fact that we save 1 byte by falling into the code to determine if we have passed an hour.
7. [Determining the current hour](#shipbell_7) - This code looks to see if the hour has changed and if so, it latches in the time based on the selected timezone.  It also calculates the number of bells and the length of the sequence necessary to play for that number of bells.
8. [Playing the next note piece](#shipbell_8) - The really tricky part here is that we have self-modifying code that generates a BRSET instruction to test the next bit in the currently selected byte.  Once we have done so, we load up a tone and stuff it into the hardware.

## More Random Numbers and Marquis - PICK6 example

Philip Hudnott <Philip.hudnott@btinternet.com> came up with this idea for a wristapp to pick lottery numbers. Overall, this is pretty simple wristapp to write, but it really showed the need for a decent random number generator. Fortunately, Alan Beale <biljir@pobox.com> provided me with a great MWC (multiply-with-carry) algorithm. Feel free to use the random number generator for other programs, it has some pretty good behavior. Overall, this program has very little changes from the 3BALL example, so getting into it should be pretty easy.

;Name: PICK6

;Version: PICK6

;Description: A sample lottery number picker to pick 6 numbers out of a pool of 49 numbers (no duplicates allowed).

; To use it, just select it as the current app and it will pick a set of 6 numbers for you. To get another set,

; just press the next button. This is for amusement only (but if you win anything because of it, I would welcome

; anything that you send me).

;

;by John A. Toebes, VIII

;

;HelpFile: watchapp.hlp

;HelpTopic: 100

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Copyright (C) 1997 John A. Toebes, VIII \*

;\* All Rights Reserved \*

;\* This program may not be distributed in any form without the permission of the author \*

;\* jtoebes@geocities.com \*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; (1) Program specific constants

;

INCLUDE "WRISTAPP.I"

;

; Program specific constants

;

RAND\_RANGE EQU 48 ; This is the number of items to select from (1 to RAND\_RANGE+1)

CURRENT\_TIC EQU $27 ; Current system clock tic (Timer)

RAND\_WCL EQU $61

RAND\_WCH EQU $62

RAND\_WNL EQU $63

RAND\_WNH EQU $64

THIS\_PICK EQU $65 ; We can share this with MARQ\_POS since we don't do both at the same time

MARQ\_POS EQU $65

TEMPL EQU $66

TEMPH EQU $67

START EQU \*

BASE\_TAB EQU $FE

;

; (2) System entry point vectors

;

L0110: jmp MAIN ; The main entry point - [WRIST\_MAIN](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_MAIN)

L0113: bclr 1,BTNFLAGS ; Called when we are suspended for any reason - [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND)

rts

L0116: jmp FLASH ; Called to handle any timers or time events - [WRIST\_DOTIC](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_DOTIC)

L0119: bclr 1,BTNFLAGS ; Called when the COMM app starts and we have timers pending - [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM)

rts

L011c: rts ; Called when the COMM app loads new data - [WRIST\_NEWDATA](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_NEWDATA)

nop

nop

L011f: lda STATETAB,X ; The state table get routine - [WRIST\_GETSTATE](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_GETSTATE)

rts

L0123: jmp HANDLE\_STATE0

db STATETAB-STATETAB

;

; (3) Program strings

;

S6\_MARQ timex6 " +O+ "

S8\_TITLE Timex " PICK-6 "

MARQ\_SEL

DB S6\_MARQ+2-START

DB S6\_MARQ+3-START

DB S6\_MARQ+2-START

DB S6\_MARQ+1-START

DB S6\_MARQ-START

DB S6\_MARQ+1-START

;

; (4) State Table

;

STATETAB:

db 0

db EVT\_ENTER,TIM2\_16TIC,0 ; Initial state

db EVT\_RESUME,TIM\_ONCE,0 ; Resume from a nested app

db EVT\_DNNEXT,TIM2\_16TIC,0 ; Next button

db EVT\_TIMER2,TIM\_ONCE,0 ; Timer

db EVT\_MODE,TIM\_ONCE,$FF ; Mode button

db EVT\_END

PICK\_VALS db 0,0,0,0,0,0,0,$FF

;

; (5) This flashes the text on the screen

;

FLASH

lda CURRENT\_APP ; See which app is currently running

cmp #APP\_WRIST ; Is it us?

bne L0113 ; No, so just turn off the tic timer since we don't need it

ldx #5

lda MARQ\_POS

jsr [INCA\_WRAPX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#INCA_WRAPX)

sta MARQ\_POS

tax

lda MARQ\_SEL,X

jsr [PUT6MID](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUT6MID)

ldx MARQ\_POS

lda MARQ\_SEL,X

jmp PUT6TOP

;

; (6) They want us to do it again

;

DOITAGAIN ; Tell them we are going to do it again

clr MARQ\_POS

bset 1,BTNFLAGS

jsr [CLEARALL](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#CLEARALL)

jmp BANNER

;

; (7) State Table 0 Handler

; This is called to process the state events.

; We see ENTER, RESUME, TIMER2 and NEXT events

;

HANDLE\_STATE0:

bset 1,APP\_FLAGS ; Indicate that we can be suspended

bclr 1,BTNFLAGS

lda BTNSTATE

cmp #EVT\_DNNEXT ; Did they press the next button?

beq DOITAGAIN

cmp #EVT\_ENTER ; Or did we start out

beq DOITAGAIN

cmp #EVT\_RESUME

beq REFRESH

;

; (8) Select a random answer

;

SHOWIT

clra

ldx #6

CLEARIT

sta PICK\_VALS-1,X

decx

bne CLEARIT

;

; We want to pick 6 random numbers. The first needs to be in the range 1 ... RAND\_RANGE

; The second should be in the range 1 ... (RAND\_RANGE-1)

; The third should be in the range 1 ... (RAND\_RANGE-2)

; The fourth should be in the range 1 ... (RAND\_RANGE-3)

; The fifth should be in the range 1 ... (RAND\_RANGE-4)

; The sixth should be in the range 1 ... (RAND\_RANGE-5)

;

clr THIS\_PICK

ONE\_MORE\_PICK

REPICK

jsr [RAND16](#PICK6_11)

and #63

sta TEMPL

lda #RAND\_RANGE

sub THIS\_PICK

cmp TEMPL

blo REPICK

lda TEMPL

bsr [INSERT\_NUM](#PICK6_9)

inc THIS\_PICK

lda THIS\_PICK

cmp #6

bne ONE\_MORE\_PICK

bra REFRESH

;

; (9) Insert a number in the list

;

INSERT\_NUM

inca

ldx #(PICK\_VALS-1)-BASE\_TAB ; Index so that we can use the short addressing mode

TRY\_NEXT

incx ; Advance to the next number

tst BASE\_TAB,X ; Is it an empty slot?

bne NOT\_END ; No, try some more

sta BASE\_TAB,X ; Yes, just toss it in there

rts ; And return

NOT\_END

cmp BASE\_TAB,X ; Non-empty slot, are we less than it?

blo PUT\_HERE ; Yes, so we go here

inca ; No, Greater than or equal, we need to increment one and try again

bra TRY\_NEXT

PUT\_HERE

sta TEMPL

lda BASE\_TAB,X

sta TEMPH

lda TEMPL

sta BASE\_TAB,X

lda TEMPH

incx

tsta

bne PUT\_HERE

rts

;

; (10) Display the currently selected random numbers

;

REFRESH

ldx PICK\_VALS

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTTOP12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP12)

ldx PICK\_VALS+1

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTTOP34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP34)

ldx PICK\_VALS+2

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTTOP56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTTOP56)

ldx PICK\_VALS+3

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTMID12](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID12)

ldx PICK\_VALS+4

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTMID34](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID34)

ldx PICK\_VALS+5

bsr [GOFMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

jsr [PUTMID56](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#PUTMID56)

lda #ROW\_MP23

sta DISP\_ROW

bset COL\_MP23,DISP\_COL

lda #ROW\_MP45

sta DISP\_ROW

bset COL\_MP45,DISP\_COL

lda #ROW\_TP23

sta DISP\_ROW

bset COL\_TP23,DISP\_COL

lda #ROW\_TP45

sta DISP\_ROW

bset COL\_TP45,DISP\_COL

BANNER

lda #S8\_TITLE-START ; And show the mode on the bottom

jmp [BANNER8](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#BANNER8)

GOFMTX JMP [FMTX](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\routines.html#FMTX)

; (11) Here is an excellent random number generator

; it comes courtesy of Alan Beale <biljir@pobox.com%gt;

; The following C code gives a good MWC (multiply-with-carry)

; generator. This type is generally superior to linear

; congruential generators. As a bonus, there is no particular advantage to using the high-order

; rather than the low-order bits.

; The algorithm was developed and analyzed by George

; Marsaglia, a very well-known scholar of random number lore.

;

; The code assumes 16 bit shorts and 32 bit longs (hardly surprising).

;

;static unsigned short wn,wc; /\* random number and carry \*/

;

;unsigned short rand() {

; unsigned long temp;

; temp = 18000\*wn + wc;

; wc = temp >> 16;

; wn = temp & 0xffff;

; return wn;

;}

;

;To seed, set wn to anything you like, and wc to anything between 0 and 17999.

;

; Translating this into assembler is

;nHnL\*0x4650 + RAND\_WCHcL

;

; unsigned long temp;

; temp = 18000\*wn + wc;

; wc = temp >> 16;

; wn = temp & 0xffff;

; return wn;

; temp = 0x4650 \* n + c

; temp = 0x4650 \* nHnL + cHcL

; temp = (0x4600 + 0x50) \* (nH00 + nL) + cHcL

; temp = 0x4600\*nH00 + 0x4600\*nL + 0x50\*nH00 + 0x50\*nL + cHcL

; temp = 0x46\*nH\*0x10000 + 0x46\*nL\*0x100 + 0x50\*nH\*0x1000 + 0x50\*nL + cHcL

; We construct the 32bit result into tH tL cH cL and then swap the 16 bit values

; once we have no more need of the original numbers in the calculation

;

RAND\_MULT EQU 18000 ; This is for the random number generator

RAND\_MULTH EQU RAND\_MULT/256

RAND\_MULTL EQU RAND\_MULT&255

RAND16

lda RAND\_WNL ; A=nL

ldx RAND\_MULTL ; X=0x50

mul ; X:A = 0x50\*nL

add RAND\_WCL ; A=Low(0x50nL)+cL

sta RAND\_WCL ; cL=Low(0x50nL)+cL

txa ; A=High(0x50nL)

adc RAND\_WCH ; A=High(0x50nL)+cH

sta RAND\_WCH ; cH=High(0x50nL)+cH

clra ; A=0

sta TEMPH ; tH=0

adc #0 ; A=Carry(0x50nL)+cH

sta TEMPL ; tL=Carry(0x50nL)+cH

lda RAND\_WNL ; A=nL

ldx RAND\_MULTH ; X=0x46

bsr RAND\_SUB ; tL:cH += 0x46\*nL tH=carry(0x46\*nL)

lda RAND\_WNH ; A=nH

ldx RAND\_MULTL ; X=0x50

bsr RAND\_SUB ; tL:cH += 0x50\*nH tH=carry(0x50\*nH)

lda RAND\_WNH ; A=nH

ldx RAND\_WCL ; X=cL

stx RAND\_WNL ; nL=cL

ldx RAND\_WCH ; X=cH

stx RAND\_WNH ; hH=cH

ldx RAND\_MULTH ; X=0x46

mul ; X:A=0x46\*nH

add TEMPL ; A=Low(0x46\*nH)+tL

sta RAND\_WCL ; nL=Low(0x46\*nH)+tL

txa ; A=High(0x46\*nH)

adc TEMPH ; A=High(0x46\*nH)+tH

sta RAND\_WCH ; nH=High(0x46\*nH)+tH

rts

RAND\_SUB

mul ; Compute the values

add RAND\_WCH ; A=LOW(result)+cH

sta RAND\_WCH ; cH=Low(result)+cH

txa ; X=High(result)

adc TEMPL ; X=High(result)+tL+Carry(low(result)+cH)

sta TEMPL ; tL=High(result)+tL+Carry(low(result)+cH)

clra ; A=0

adc TEMPH ; A=carry(High(result)+tL+Carry(low(result)+cH))+tH

sta TEMPH ; tH=carry(High(result)+tL+Carry(low(result)+cH))+tH

rts

;

; (12) This is the main initialization routine which is called when we first get the app into memory

;

MAIN:

lda #$c0 ; We want button beeps and to indicate that we have been loaded

sta WRISTAPP\_FLAGS

lda CURRENT\_TIC

sta RAND\_WNL

sta RAND\_WNH

sta RAND\_WCL

and #$3f

sta RAND\_WCH

rts

1. [Program specific constants](#PICK6_1) - We have several variables - RAND\_WCL, RAND\_WCH, RAND\_WNL and RAND\_WNH which we use for the random number routine. CURRENT\_TIC is what is set by the system when it reads the clock to keep the watch time up to date.  We use it once to provide a seed for the random number generator. Note that we are overlapping the use of THIS\_PICK and MARQ\_POS to save one byte of low ram.
2. [System entry point vectors](#PICK6_2) - identical to the 3BALL example, This one gets to be a little fun.  Notice for the [WRIST\_SUSPEND](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_SUSPEND) and [WRIST\_INCOMM](file:///\\Server\Users\JOHN\Visual%20Studio%20Projects\Datalink\FrameWeb\wristappformat.html#WRIST_INCOMM) routines that we don't have a JMP instruction, but instead put the actual code in line.  This saves use a couple of bytes.
3. [Program strings](#PICK6_3) - We are pretty frugal here in reusing blanks at the end of the string very liberally.  Also note the S6\_MARQ string which has blanks at the start and end so that it can shuffle left and right on the display but always have blanks visible.  The MARQ\_SEL and MSG\_SEL tables are simply offsets that allow us to select the message with a simple load instruction instead of having to calculate the offset.
4. [State Table](#PICK6_4) - This is pretty vanilla here except for the fact that we have a very long time interval after the DNNEXT and ENTER events.  It is during this time that the Marquis runs.  We could make it even longer, but this seems to be a good compromise between seeing something happen and actually getting a result in a reasonable time.
5. [State Table 0 Handler](#PICK6_5) - Extremely simple, there are only four events that we want to see and this is the typical test and branch one.  The only unique thing here is that we turn off the Marquis timer as soon as we get any event.
6. [This flashes the text on the screen](#PICK6_6) - This is the cheap way to do a Marquis.  Just have a string wider than the display and change the offset from the start at which you start to display.  For this one, there are only 6 states and we select the starting offset from the table based on our current cycle.  Note that this routine is called by the TIC timer which is enabled when they want a new random number.  Eventually the timer for the main event will run out and they will simply stop calling us.
7. [They want us to do it again](#PICK6_7) - Whenever we want to do a new random number, we just start the Marquis tic timer and set up the display.
8. [Select a random answer](#PICK6_8) - This is really the meat of this wristapp. We need to pick 6 random numbers and sort them. Fortunately, we can take advantage of the sorting as part of our random number selection.
9. [Insert a number in the list](#PICK6_9) - Given a random number, add it to the list of random numbers in sorted order. Essentially, we start at the beginning of the list and go until we either find a slot where we need to insert the number in order or we hit the end of the list. If we hit the end of the list, we store the number there and return. Otherwise we insert the number at the appropriate spot. One additional thing that we do is increment the number by 1 for each entry in the that is less than it. It makes sense, but you need to think about why this works.
10. [Display the currently selected random numbers](#PICK6_10) - Given the 6 random numbers, we just put them on the display separated by periods. Note the series of BSR instructions to the GOFMTX label. Since there were 6 calls to it, we were about to reduce the 6 3-byte instructions to 6 2-byte instructions plus one 3-byte instruction to do the call for a savings of 3 bytes.
11. [Here is a random number generator](#PICK6_11) - This is great random number generator that you might want to grab for any other code that you might write.
12. [This is the main initialization routine which is called when we first get the app into memory](#PICK6_12) - Very boring stuff here, but we do take a moment to initialize the random number seed with the current tic count just to make it a little more variable.

# Index

1

150, 6, 8, 12, 24, 30, 31, 33, 38, 64, 65

150s, 6, 8, 24, 30, 31, 33, 38, 64, 65

3

3Ball, 65, 131

6

6805, 23, 24, 30

7

75, 6

A

Adjustable Timer, 6

Applet file header, 33

ASM6805, 30, 100

assemblers, 30

C

Character Set, 12

Conversion, 7

CopyMe, 7

CPACKET\_ALARM, 10

CPACKET\_BEEPS, 12

CPACKET\_CLEAR, 10

CPACKET\_DATA, 9, 10

CPACKET\_END, 10

CPACKET\_JMPMEM, 8, 9

CPACKET\_MEM, 8, 11

CPACKET\_SECT, 9, 10

CPACKET\_SKIP, 9

CPACKET\_START, 8, 24

CPACKET\_TIME, 11

CRC, 8, 9, 10, 11, 12, 33, 34, 63

D

DAYFIND, 84, 85, 87

Display, 12

Download Protocol, 7

E

EEPROM, 9, 10, 24, 45, 46, 47, 48, 64, 107, 108, 109, 111, 114

ENDOFF, 96

EVENT\_END, 35

EVT\_ANY, 36

EVT\_ANY4, 36

EVT\_END, 37, 68, 70, 73, 76, 80, 86, 94, 97, 101, 108, 109, 118, 132, 136

EVT\_ENTER, 35, 37, 67, 70, 72, 73, 76, 77, 80, 86, 93, 94, 97, 101, 108, 109, 118, 120, 132, 136

EVT\_GLOW, 36

EVT\_IDLE, 36

EVT\_MODE, 36, 68, 70, 73, 76, 80, 81, 86, 88, 93, 97, 101, 104, 108, 112, 118, 120, 132, 136

EVT\_NEST, 35, 37

EVT\_NEXT, 36, 73, 77, 81, 88, 104, 112

EVT\_PREV, 36, 73, 77, 81, 88, 104, 112, 120

EVT\_RESUME, 35, 37, 67, 70, 73, 76, 80, 86, 93, 94, 97, 101, 108, 109, 118, 132, 136

EVT\_SET, 35, 36, 73, 77, 80, 86, 101, 108, 118, 120

EVT\_TIMER1, 36, 37, 86

EVT\_TIMER2, 36, 37, 70, 73, 76, 80, 86, 88, 93, 94, 101, 102, 108, 109, 110, 118, 132

EVT\_USER0, 36, 37, 94, 95, 101, 108

EVT\_USER1, 36, 37

EVT\_USER2, 36, 37, 80, 81, 86, 88, 101, 104, 109, 112, 118, 120

EVT\_USER3, 36, 37, 86, 88, 89

G

Golf, 7

H

HexDump, 64, 100, 107

M

Melody Tester, 6

N

Note, 6, 8, 11, 12, 17, 25, 26, 32, 38, 46, 51, 60, 63, 72, 92, 94, 99, 103, 106, 110, 114, 117, 121, 134

P

Packet Format, 8

PASSWD, 78, 79

Preset Timer, 6

Pulse, 7

S

Ships Bells, 135

Sound Hardware, 25

sound scheme, 9, 26, 32, 65, 92, 129, 135

Sound Scheme, 26, 129

Sound Sequence, 26

Soundlet, 26

Spend Watch, 114

State Table, 32, 35, 36, 67, 68, 69, 70, 71, 72, 73, 75, 76, 78, 80, 81, 83, 84, 86, 87, 92, 93, 94, 95, 96, 97, 99, 101, 104, 106, 107, 108, 109, 112, 114, 120, 128, 132, 134, 136, 138

Stopwatch, 6

Strings, 33

Sync Bits, 7

Sync bytes, 7

Synchronization, 7

System Sound, 26

T

TestSnd, 92

TIMEX6, 13

Tone, 25, 26

W

Week of the Year, 6

World Time, 7

WRIST\_DOTIC, 32, 34, 67, 69, 72, 76, 79, 85, 93, 96, 98, 99, 100, 108, 116, 131, 135, 137

WRIST\_GETSTATE, 32, 67, 70, 73, 76, 79, 85, 93, 96, 100, 108, 116, 131, 135

WRIST\_INCOMM, 32, 67, 69, 72, 76, 79, 85, 93, 96, 99, 100, 108, 116, 131, 134, 135, 138

WRIST\_JMP\_STATE0, 32

WRIST\_JMP\_STATE1, 32

WRIST\_MAIN, 32, 53, 54, 67, 69, 72, 76, 79, 85, 93, 96, 100, 108, 116, 131, 135

WRIST\_NEWDATA, 32, 67, 69, 72, 76, 79, 85, 93, 96, 99, 100, 108, 116, 131, 135

WRIST\_OFF\_STATE0, 32

WRIST\_OFF\_STATE1, 32

WRIST\_SUSPEND, 32, 35, 67, 69, 72, 76, 79, 85, 93, 96, 100, 108, 116, 131, 134, 135

WRISTAPP\_FLAGS, 34, 69, 71, 72, 74, 75, 78, 82, 84, 91, 95, 96, 98, 99, 106, 113, 127, 134, 137

Z

ZAP, 24, 31, 33, 63, 64

ZSM, 31, 100